DEVELOPING A VALIDATION METHODOLOGY FOR TACAIR SOAR AGENTS IN EAAGLES

GRADUATE RESEARCH PROJECT

Lewis E. Alford III, Maj, USAF

Brian A. Dudas, Maj, USAF

AFIT/GOS/ENS/05-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOS/ENS/05-01

DEVELOPING A VALIDATION METHODOLOGY FOR TACAIR SOAR AGENTS IN EAAGLES

GRADUATE RESEARCH PROJECT

Presented to the Faculty

Department of Engineering Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Operational Sciences

Lewis E. Alford III, BS

Maj, USAF

Brian A. Dudas, MS

Maj, USAF

May 2005

AFIT/GOS/ENS/05-01

DEVELOPING A VALIDATION METHODOLOGY FOR TACAIR SOAR AGENTS IN EAAGLES

Lewis E. Alford III, BS

Maj, USAF

Brian A. Dudas, MS

Maj, USAF

Approved:

______________________________ __________
Dr. John O. Miller (Chairman) Date

AFIT/ENS/GOS/05-01

Abstract

The Air Force Simulation and Analysis Facility (SIMAF) at Wright Patterson AFB is currently conducting an effort to integrate TacAir Soar agents into the Enhanced Air-to-air and Air-to-Ground Linked Environment Simulation (EAAGLES) environment. SIMAF plans to support customizable agents in the Soar interpretive language, compilable for use in EAAGLES. TacAir Soar (TAS) agents are currently only utilized in the Joint Semi-Automated Forces (JSAF) environment, but have potential for use in EAAGLES. SIMAF requested research be conducted on a validation methodology to apply to the agents' behavior once they have been successfully imported into the EAAGLES environment. This methodology developed could then be used on all variants of the final version of the agents, once delivered by Soar Technologies, Inc. This research project investigated the possibility of "verification by comparison" with the TacAir Soar agent's current behavior and performance in JSAF while integrated with "human in the loop" simulators. By comparing the new models to the already validated models in JSAF, the desire was to eliminate the need for a "ground up" verification. This paper outlines the steps taken to successfully construct a working JSAF/TacAir Soar model, as well as illustrates the deficiencies and limitations encountered. Finally, recommendations are made as to the future development of validation/verification methodology for SIMAF's efforts.

Acknowledgments

We would like to express our sincere appreciation to our faculty advisor, Dr. John O. Miller, for his guidance and support throughout the course of this research effort. His insight and experience was certainly appreciated. We would also like to thank Mr. Michael Hass, from the Air Force Research Laboratory for the unending support provided to us in this endeavor.

Brian A. Dudas

Lewis E. Alford, III

Table of Contents

List of Figures

Page

List of Tables

Page

DEVELOPING A VALIDATION METHODOLOGY FOR TACAIR SOAR AGENTS IN EAAGLES

I. Introduction

Background

This project was sponsored by the Air Force Simulation and Analysis Facility (SIMAF) at Wright Patterson AFB, OH. SIMAF is currently conducting an effort which integrates TacAir Soar with Enhanced Air-to-air and Air-to-Ground Linked Environment Simulation (EAAGLES). SIMAF plans to provide customizable agents in the Soar interpretive language, compilable for use in the EAAGLES environment. The initial project concept was for the development of both friendly and opposing force (OPFOR) agents that would replicate the behavior and tactics of modern-day air forces. These agents were to be developed in TacAir Soar via the Soar interpretive language and then imported into EAAGLES.

Initial Review

After an initial review of the composition of the TacAir Soar agent models, we discovered that the current agents were highly developed intelligent agents. Their logic invokes over 9000 rules, which have been developed through interaction with military subject matter experts (SMEs) since the inception of the project in 1994. After installing the Soar Interpretive Language and development kit, and constructing several programs to familiarize ourselves with the language, we learned that the TacAir Soar agents included with JSAF were only the compiled binaries, not the source code. Soar Tech, Inc. maintains the ownership rights to the source code, and SIMAF was in the process of completing an agreement which would give SIMAF the rights to an updated version of TacAir Soar along with the most up-to-date agents. We were not therefore able to directly view the rules used to build the agents to determine their validity, nor were we able to add or modify the agents' behavior and rules.

TacAir Soar agents have existed in JSAF since 1994, and have undergone multiple levels of validation and verification by multiple agencies. Their performance is well-documented in several large exercise summaries such as *RoadRunner, Coyote,* and *Flight Lead Upgrade*. With this in mind, we decided that it would be much more feasible to benchmark the agents' behaviors in their current JSAF environment, using that to verify that their behaviors don't change once the agents are imported into EAAGLES. Since there are so many varieties of agent roles and behaviors in the TacAir Soar model, we elected to narrow the scope of our investigation to a single type of agent in a single role. Our intent was to perform a statistical analysis of JSAF TacAir Soar agents operating in a wingman role, with a "human-in-the-loop" flight lead, while performing a sampling of mission-critical tasks. This scenario would be easily repeatable in the EAAGLES environment with SIMAF's current simulation system. We planned to make a detailed comparison of the results of the scenarios as obtained from each environment to determine whether or not the agents' behaviors remained consistent after being imported into the EAAGLES environment. It is important to emphasize that we did not plan to *validate* the TacAir Soar agents, but merely to *verify* the consistency of their behavior and performance between the two environments.

Unfortunately, the contracting process between SIMAF and Soar Technologies, Inc. was not completed prior to the commencement of our study. In addition, Soar Technologies had not yet completed the most current version of their TacAir Soar agents that were planned for purchase by SIMAF. These two factors left us not only with the inability to examine the TacAir Soar models in EAAGLES, but also with no way to begin testing the precise performance of the final version of the agents that were to be delivered.

Research Focus

Having exhaustively searched and compiled a list of the resources at our disposal, we determined that the best course of action was for us to simply study the current agents' behavior in JSAF and provide SIMAF with a verification methodology to apply to the agents' behavior once they have been successfully imported into the EAAGLES environment. The methodology developed could then be used on all variants of the final version of the agents, once delivered by Soar Technologies, Inc., during SIMAF's EAAGLES conversion process.

II. Literature Review

Chapter Overview

The purpose of this chapter is to provide a thorough review of the relevant components of the project. First, this chapter provides an in-depth review of the Soar interpretive language and TacAir Soar agents, their history, and utilization in successful systems, programs, and major exercises. Next this chapter presents a background on JSAF, its history, and current scope of utilization. Finally, this chapter outlays the history, capabilities, and flexibility of EAAGLES.

SOAR Background

The Soar interpretive language was developed as an architecture for constructing general intelligent systems. It was initially designed at the University of Michigan and has been in use since 1983. Specifically, it was intended to be used to build systems that work on the full range of tasks expected of an intelligent agent, from highly routine to extremely difficult, open-ended problems. Its agents are able to interact with the outside world and learn about all aspects of the tasks and their own performance of those tasks. Its agents are unique in that they are goal-based. In Soar, there is a single representation of permanent knowledge (productions), a single representation of temporary knowledge (objects with attributes and values), a single mechanism for generating goals (automatic subgoaling), and a single learning mechanism. Additionally, all decisions are made in parallel through the combination of relevant knowledge at runtime. In Soar, every decision is based on the current interpretation of sensory data and any relevant knowledge retrieved from permanent memory. Decisions are never precompiled into uninterruptible sequences (Laird, 2004:1).

Soar agents and their associated environment share many characteristics of human thinking and our real-world environment. First, the agents are goal-oriented (Lehman, 2005:5). Soar agents are also able to identify sub-goals and tasks which must be accomplished as well in order to meet the primary goal. Second, the soar architecture reflects a rich, complex, detailed environment (Lehman, 2005:6). Third, Soar agents are flexible and function in step with their environment. They have the ability to take appropriate actions as a result of unexpected events very similarly to the way that humans do.

TacAir Soar was developed between 1992 and 1997 at the University of Michigan (UoM). That same development team then left UoM and formed Soar Technology, Inc. Like the original Soar, TacAir Soar was originally designed to be a fully autonomous intelligent agent system. Its specialty was to provide high-fidelity, realistic, entity-level behaviors for a wide range of aircraft and missions (History, 2005:1). These missions/tasks included but were not limited to basic formation, air-to-air, and air-to-ground missions. These agents have been used in various mission training exercises including those held at the U.S. Air Force Warfighter Training Laboratory in Mesa, AZ.

One example of TacAir Soar's use was in a 1997 exercise named STOW-97 (Synthetic Theatre of War). During this exercise, all U.S. aircraft were modeled as TacAir Soar entities. Over 722 flights were "flown" during the 48 hour duration of the exercise. Results included a 99% mission launch rate and 94-96% of the missions flew without errors. At that time the agents had approximately 5,500 rules. They were dynamically controlled using a "SoarSpeak" module, but this was used to provide only minimal oversight. For the most part the agents used autonomous decision-making in their operations (Kalus, 2003:74). Because of the flexibility of platforms available which can run TacAir Soar, approximately 30 PCs were used to support 4-6 aircraft each for a total of approximately 100 aircraft airborne at any one time (Kalus, 2003:75).

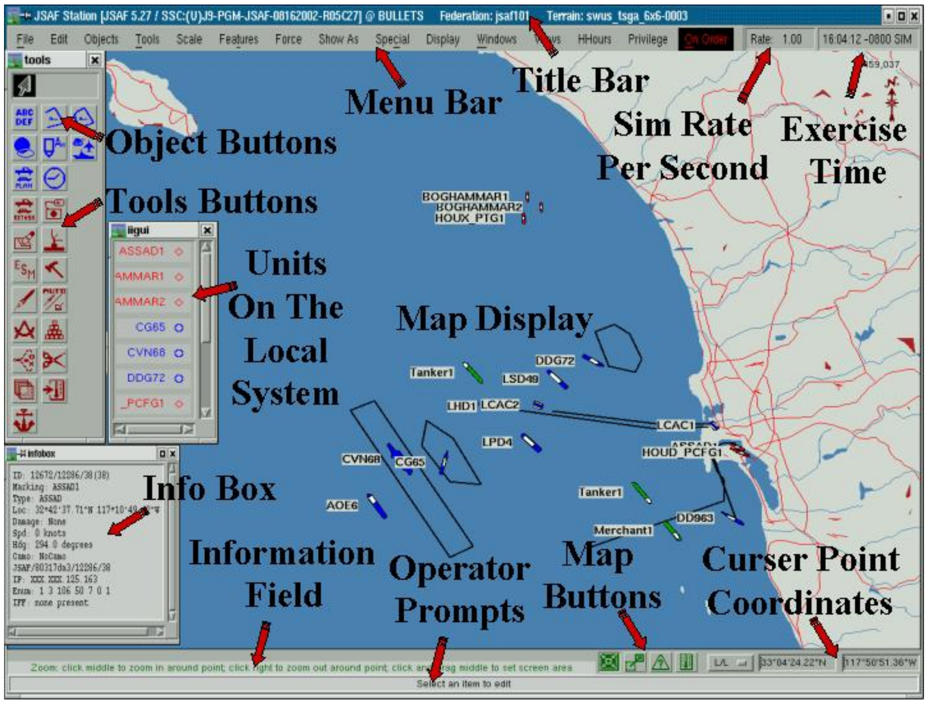
Similar exercises were held in 1998, named *Coyote* and *RoadRunner*. These exercises once again tasked TacAir Soar to provide computer generated agents not only to operate in a synthetic environment with other agents but also to interact with "humans in the loop," USAF pilots operating in F-16 and A-10 simulators (Nielson, 2005: sec. 3). In addition, these exercises tested TacAir Soar's ability to work in a networked environment comprising seven different locations across the United States. During the *Coyote '98* exercise, one of the major tasks was to test the ability of TacAir Soar agents to interact with humans in the simulations. The agents were required to accomplish all normal wingmen tasks (correct formation position, contract adherence, etc.) and to communicate with their human flight leads and air controllers. To accomplish this in TacAir Soar, commercial off-the-shelf software was used to convert the flight leads' and controllers' voice directives to text messages which were then passed over simulated radios. Responses were passed along these same radios and synthesized to speech, so the flight leads and controllers could hear the responses over their headsets. The specialization of this system for the military air domain is called SoarSpeak (Nielson, 2005: sec. 5.1). The agents had a limited vocabulary of voice commands that they could interpret and understand. With literally five minutes training in the vocabulary, a new pilot was able to lead TAS-controlled aircraft into combat (Nielson, 2005: sec. 5.2). One interesting scenario developed which highlighted the agents' vocabulary limitations. As the first pilot was attempting to "control" his wingman, his lexicon was less than precise. Consequently, the TAS wingman moved out of visual range of his lead aircraft. When flight lead (the human pilot) directed him to turn to heading 270, the TAS aircraft responded "Roger, authenticate XYZ" while maintaining his current vector. The TAS wingman was complying with theatre procedures that required him to verify unknown directives with coded authentication procedures. When the lead pilot authenticated accurately, the TAS wingman immediately followed lead's directions, and successfully rejoined the number 1 aircraft (Nielson, 2005: sec. 5.2). TacAir Soar agents performed well during these exercises and they were considered successful.

Joint Semi-Automated Forces Introduction

The Joint Semi-Automated Forces (JSAF) is a computer generated forces (CGF) system that is used by the U.S. Joint Forces Command for joint experimentation, by the U.S. Navy for Fleet Battle Experiments, and by the AFRL Human Effectiveness Directorate (AFRL/HEA) in support of the Distributed Mission Training (DMT) program (Trevisani,2003:1). The JSAF modeling and simulation program is used to generate entity level units such as tanks, ships, aircraft, and individual combatants, including their associated sensor systems and munitions (Joint, 2003:1). JSAF runs on PC's under the Linux operating system and is also supported on Sun, SGI, and IBM hardware. It is easily ported to most versions of Unix (Joint, 2005b:1). Unfortunately, it is not supported in any type of Windows environment. Due to its foundations in Linux, many different versions of JSAF may be found in the modeling and simulation community. Additionally, each upgrade to JSAF requires several different “builds” to be compatible with different versions of Linux.

JSAF began life as the Synthetic Theater of War (STOW) and was created by DARPA as an Advanced Concept Technology Demonstration (ATCD) (Joint, 1998:1). It is capable of running in a standalone environment on a single PC or as a distributed simulation in a network communicating via the Higher Level Architecture (HLA) or Distributed Interactive Simulation (DIS) protocols. The original concept of its development was to provide enhanced simulation fidelity based on combat resolution at the weapons system level; realistic simulation of command and control behavior; networking and information flow technology; and the capability to provide knowledge-based autonomous forces in simulation with man-in-the-loop participation wherever desired. It was initially tested in October of 1997 at Unified Endeavor 98-1, a Joint Task Force level exercise (Joint, 2005a:1). It saw its first large-scale usage as part of the *Coyote* exercise hosted by AFRL, Mesa.

Currently, JSAF provides a synthetic environment in a representation of real world terrain, oceans, and weather conditions that affect the behaviors and capabilities of the synthetic forces (Joint, 2003:1). JSAF users control automated behavior with tasks and task frames where task frames are collections of tasks that run simultaneously (Joint, 2003:2). JSAF provides a Windows-type graphic user interface as seen in Figure 1.



```
ID: 12672/12286/38(38)
Marking: ASSAD1
Type: ASSAD
Loc: 32*42'37.71"N 117*10'49.[illegible]"W
Damage: None
Spd: 0 knots
Hdg: 294.0 degrees
Camo: NoCamo
JSAF/80317da3/12286/38
IP: XXX.XXX.125.163
Enum: 1 3 106 50 7 0 1
IFF: none present
```

Zoom: click middle to zoom in around point; click right to zoom out around point; click and drag middle to set screen area

Select an item to edit

Figure 1. JSAF Screenshot

In addition to its semi-automated forces, JSAF also supports reduced resolution models for the simulation of entities requiring less intelligence/combat capability; e.g. marshalling forces, civilians, units in transit, etc. This allows significant scaling; JSAF exercises routinely run over 30,000 entities at a time and it appears 100,000 is easily within the realm of the possible depending on need (Joint, 2005b:1). Additionally, JSAF uses a "gateway" to communicate in DIS with those models and/or systems which are incapable of HLA communications. Users may control JSAF behavior by: (1) creating pre-planned missions, (2) setting up reactions, and (3) issuing immediate commands (Joint, 2003:2).

The current office of primary responsibility for JSAF is The U.S. Joint Forces Command (JFCOM) Joint Experimentation Directorate, J9. Their website can be found at: http://www.jfcom.mil/about/abt_j9.htm.

Enhanced Air-to-air/Air-to-Ground Linked Environment Simulation Introduction

The Enhanced Air-to-air and Air-to-Ground Linked Environment Simulation (EAAGLES) is a simulation environment specially tailored to the complexities of the air warfare environment. The EAAGLES program began in June 2002 with the first prototype delivered in December of that year (Menke, 2003:12). The primary goal in the development of EAAGLES was to produce a simulation environment capable of providing the high-fidelity modeling of new technology that was not and is not available in the current USAF model toolkits. EAAGLES is designed to significantly reduce acquisition cycle time by addressing current modeling and simulation infrastructure limitations (Menke, 2003:14). Many of the models currently in use were initially designed and used for surface combatant simulations and lack the capability to replicate the real-time, dynamic environment of air combat. While some of them have been adapted to simulate airborne entities, most only model aircraft in a support-type role. EAAGLES was designed to support thousands of “Players” in a modular fashion. This lends itself to realistic modeling of not only the air combat portion of the fight but also the air defense systems, which can be quite intricate in their own right (Menke, 2003:6).

SIMAF’s ultimate goal is to use EAAGLES along with the existing capabilities in the test community, to reduce the impact of existing test program deficiencies in threat density and interoperability applications (Levine, 2003:20). EAAGLES was designed not to replace but to augment Developmental and Operational Testing. It consists of configurable and extendable simulation components and applications that allow users to configure their EAAGLES simulations to meet their own unique requirements, which may include integration with other legacy models (Enhanced, 2004b). It was designed to support rapid reconfiguration of modeled systems and to do so while simulating 500 air entities alone. It can support hundreds of other players and do so while providing real-time performance (Enhanced, 2004a). Because of the robustness of the EAAGLES interface and this real-time capability, it is able to support “human-in-the-loop” operations, thereby incorporating the “Ultimate Agent” for any simulation.

In order to enhance its operability, EAAGLES was designed to be run on personal computers (PC) in either a Linux or Windows environment. It is capable of being integrated with dedicated servers for high fidelity in addition to being able to communicate with other models using either DIS or HLA (Menke, 2003:9). It can be run on one computer or on a network of many PCs. Additionally, a graphics toolkit is provided for modeling interactive pilot-vehicle interfaces (PVI) and control displays. These toolkits are built on the EAAGLES basic object system and can be used independently or in other simulation models (Enhanced, 2004b).

In order to more fully exploit the strengths and capabilities of the EAAGLES environment, SIMAF hopes to integrate the modular capabilities of TacAir Soar agents into EAAGLES. By maintaining a database of the rules that power the different varieties of agents, SIMAF hopes to enable different organizations and units the ability to integrate their own specific rules, roles, and missions into the current 9,000 rule agents. By fusing the parallel decision processing capabilities of the agents in a specialized role with the modular expandability of EAAGLES, SIMAF aims to provide a more realistic simulation and training environment than is currently available anywhere in the Air Force.

Summary

This chapter has provided the background of the major components involved in this research effort. First it discussed the Soar interpretive language and its framework used to construct effective TacAir Soar agents. Next, it outlined the history and current status and utilization of JSAF in JFCOM's warfighter modeling. Finally, it explained the background, capabilities, and potential of the EAAGLES system. Having fully explained these components, the report now moves on to describing the process of scenario development for the research effort.

III. Scenario Development

Chapter Overview

The process of establishing a functional system and developing a scenario for the verification was extremely involved. This chapter outlines the setup of both the hardware and JSAF/TacAir Soar software systems, and the difficulties and constraints that were encountered in that processes. It then discusses the interfacing of JSAF/TacAir Soar with EAAGLES software, and the lessons learned therein. Finally, it outlines the simulation scenario and testing processes that were planned for use in the verification effort.

Hardware and Software System Setup

The combination of Linux and JSAF 2004 made setting up the system a very difficult task. Our team consisted of two individuals quite experienced with computers in the Windows environment, both adept in several programming languages, and one having previous experience in the Unix environment. Despite this experience level, it took a total of six weeks of daily trial and error to finalize our JSAF system. The components we initially felt were necessary for the project were:

- Functioning Linux Operating System
- JSAF Simulation Environment
- TacAir Soar Agent interfaces
- Situational Awareness Panel (SAP)
- Functioning DIS Gateway for interfacing with EAAGLES
- Functioning HLA for connection to other JSAF machines

Our goal was to set this software up on a Dell M50 Precision Laptop with a Pentium 4, 3.0 GHz processor with 1GB RAM. This would allow us the ability to connect to different simulation networks at various locations, to include networks at AFIT, SIMAF, AFRL, and our test location.

We obtained three versions of JSAF:

- JSAF 2004: The most up-to-date JSAF version in use by JFCOM at this date.
- JSAF 5.32 Gold: The most recent working version of JSAF in use by AFRL/HECP.
- JSAF SAGIS: A release provided to us by Soar Tech, Inc.

We attempted installation each of these versions on each of five different versions of Linux, with the following results:

Table 1. Linux-JSAF Combinations

Functioning Components in Various Installs			
	JSAF SAGIS	JSAF 5.32 Gold	JSAF 2004
Redhat 8.0	-	JSAF, DIS, HLA	JSAF, DIS, HLA
Redhat 9.0	-	JSAF, DIS, HLA	JSAF, DIS, HLA
Fedora Core 1	-	-	JSAF, DIS
Fedora Core 2	-	-	JSAF, DIS, HLA
Fedora Core 3	-	-	**JSAF, DIS, HLA, TAS, SAP**

We experienced extreme difficulty matching the provided JSAF versions with the correct versions of C++ language compilers, HLA Run Time Infrastructure (RTI) architecture files, X-Motif files, and various other system-specific software. Only the combination of Fedora Core 3 and JSAF 2004 provided all the necessary components to continue our study.

There is a sore lack of high-level documentation on the installation procedures for JSAF. Most of the install help files are outdated, referencing 6-7 year old versions of Linux which are no longer supported. Most help files we read were in the form of informal text files geared towards experienced Linux System Administrators, not for novice Linux personnel.

We requested assistance from technical experts at the Naval Warfare Center, AFRL Mesa, JFCOM/J9, and even SoarTech, Inc. There was little to no technical support available from most of these agencies, and most assistance provided was as a matter of a "personal favor."

- JFCOM stated that they would give us whatever help they could via phone and email, but any software updates or patches required to make the software function correctly was not possible. JFCOM's current funding is focused on the Special Operations Forces (SOF) agents, and there is no funding available for TAS agent repair.
- AFRL Mesa no longer uses JSAF or TAS agents. Mr. Don Smoot, a Lockheed contractor and one of the key coordinators of the *RoadRunner* and *Coyote* exercises, is still employed there. Mr. Smoot prompted SoarTech, Inc. to provide us with one of the necessary software repair patches.
- Naval Warfare Center had no personnel actively using or familiar with TAS agents.
- Our team traveled to SoarTech, Inc.'s site in Ann Arbor, Michigan after 3 weeks of installation attempts, taking the actual hardware on which we had been

attempting to construct the system. Their software experts were unable to repair the installation to create a successful system per our specifications.

Trial and error brought us to the final software configuration we required. There is no doubt that personnel already experienced in Linux (and simulation software) could have completed this faster, and probably could have found ways to make the other 14 JSAF/Linux combinations more operational. This is one of the drawbacks that we will identify in the lessons learned section, the high level of expertise and specialization required to install and use JSAF.

Appendices A and B provide those attempting to build a working model with explicit instructions as to how to construct a system with the systems listed above to the functional level. It will hopefully allow the user to bypass the difficulties we encountered in arriving at a system capable of meeting the requirements for this tasking.

Interaction with EAAGLES

The primary goal of our project involved a functioning JSAF workstation interacting with a “human-in-the-loop” simulator working in the EAAGLES environment. AFRL/HECP possesses an operational 160° Field of View (FOV) simulator working in the EAAGLES environment, complete with terrain files, audio, and radio communication. This system, unlike the EAAGLES system at SIMAF, is fully unclassified, driving us to perform our tests at the AFRL site. Indeed, this project would have been far less productive without the great assistance of the personnel at the AFRL facility.

Although EAAGLES is designed to communicate via HLA, we did not achieve a successful HLA interface from EAAGLES to JSAF. After the difficulties we experienced with the delicate nature of the JSAF 2004 HLA RTI during system setup, we did not have the knowledge or support to attempt to construct an HLA bridge. The conflicts and differences between the different versions of the RTI between JSAF and EAAGLES could not be resolved. We were able to achieve a very stable link through DIS, using the JSAF DIS Gateway module. We later successfully ran the DIS gateway on a separate JSAF workstation to free up resources on the TAS agents machine.

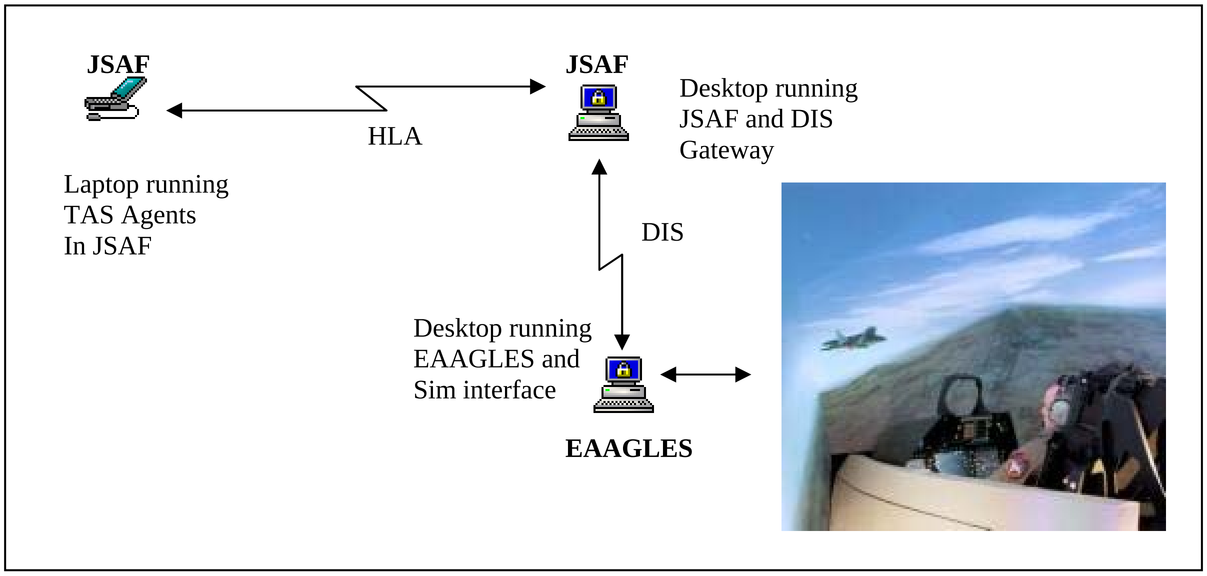

JSAF
JSAF
Desktop running JSAF and DIS Gateway
HLA
Laptop running TAS Agents In JSAF
DIS
Desktop running EAAGLES and Sim interface
EAAGLES

Figure 2. Final JSAF / EAAGLES Interface

Overall, we found the data transfer rate to be acceptable across the DIS Gateway. Since DIS performs "dead reckoning" of entities, an extended period of time without an update of a maneuvering entity made the entity appear to "jump" when the next update arrived. This was especially noticeable for an aircraft in close proximity to the human pilot in the EAAGLES simulator. By setting a high DIS update rate in the Gateway, and limiting the number of entities in our scenarios, this “jumpiness” did not cause significant distraction to the pilots in the simulator.

Some tactical information was lost in the DIS link. Aircraft successfully “killed” by the human EAAGLES pilot were not destroyed in JSAF. A purple explosion symbol occurred in JSAF to inform the user that a “hit” had occurred. This required the controller to manually remove the “killed” entity upon observing the “hit” symbology. Due to persistent objects in DIS, often times killed or removed aircraft would remain visible in the EAAGLES simulator for some time (up to 30 seconds in our trials).

The 160° FOV simulator did not afford the pilot the luxury of consistently visually monitoring his wingman. When aircraft are flying formation, most typical formation positions will place the wingman on or behind an imaginary line extending out perpendicular to the flight lead’s flight path (referred to as the “3/9 line”). Through good communication with the console controller, when the wingman was behind the flight lead’s 3/9 line (where he should be), though he was not visible, the flight lead was able to keep situationally aware of the wingman’s position.

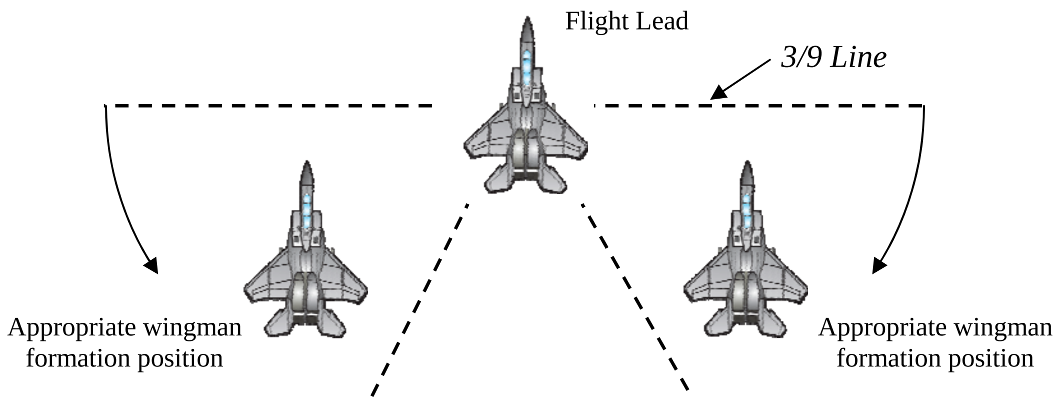

Flight Lead
3/9 Line
Appropriate wingman formation position
Appropriate wingman formation position

Figure 3. General Formation Zones

The software for emulating fighter aircraft in AFRL's unclassified simulator is extremely basic. Although there is a RADAR scope to "break out" formations at distances of up to 40 NM, there is no way to control the RADAR. There is a "super-lock" mode which allows you to achieve a lock, and toggle between targets on your scope, represented only by a target designator in the HUD. The aircraft can then fire notional radar-guided missiles at the targets. Since this is an unclassified system, these limitations did not significantly impact our study. We realize that the final verification will be performed on the classified EAAGLES system located at SIMAF, which will eliminate these constraints.

Due to the lack of functionality of the SoarSpeak module, the flight lead was unable to communicate directly with his wingman. This resulted in the only method of communication to the wingman being the Communication Panel module. The limited number of preprogrammed radio calls available in this module meant that the flight lead was unable to tactically order or target his wingman into any hostile groups. Due to the wingman's subordinate relation to a flight lead, the wingman's rules will allow him to follow only a subset of those rules. The exhaustive list of commands available is listed in Table 2; commands that the wingman will respond to or follow are annotated by a "yes" in the "Autonomous Wingman Response" columns. None of the commands enable the JSAF console operator to give a tactical order to the wingman (as if it came from the wingman's flight lead). Since the TAS agent rules for wingman seem to be that wingman agents will not actively engage bandits until targeted by their flight leads (when they are in formation with a flight lead) the agent wingmen never actually employed ordnance in any of our runs.

Table 2. Available TacAir Soar Tactical Radio Commands

Command	Autonomous Wingman Response	Command	Autonomous Wingman Response
Change attack target		Vector and escort bogey	
Say weapons load	Yes	Vector and kill bogey	
Hold at position		Broadcast report	Yes
Disregard	Yes	Close report	Yes
Vector and kill bogey		Weapons status	Yes
Vector BARCAP dir		Confirmed bandit	
Change cap length dir		Confirmed hostile	
Vector BARCAP point		Confirmed friendly	
Vector BARCAP latlong		Kill bogey	
Vector for bogey		ID bogey	

Lastly, the process of making the TAS agent believe the human EAAGLES pilot was his flight lead took some finesse. We designed a basic JSAF scenario with a 2-ship of TAS agent F-15C aircraft, callsigns *POLO1* for the flight lead and *POLO2* for the wingman, and gave them a simple Defensive Counter Air (DCA) mission to perform Combat Air Patrol (CAP) at a specific point. We armed both aircraft with AIM-9 Sidewinders (heat seeking), AIM-7 Sparrow (Semi-active RADAR guided), and AIM-120 AMRAAM (Active RADAR guided) missiles. We loaded and started this scenario in JSAF, and waited for the formation to takeoff and begin its mission. We then initiated the EAAGLES simulation, with a single aircraft flown by the human pilot, also with callsign *POLO1*. This aircraft was initiated within 3-5 miles of the TAS agents. As soon as the EAAGLES aircraft appeared on the JSAF screen, we manually deleted the JSAF (Soar agent) *POLO1*. Since there was an aircraft named *POLO1* within visual range of

POLO2, the wingman did not differentiate between the two other aircraft, and assumed that the single remaining *POLO1* was his flight lead. He therefore immediately maneuvered into formation, and maintained the Soar rules of wingman performance throughout the remainder of the sortie run.

JSAF – EAAGLES TAS Agent Transition Verification

The primary goal of this study was to provide a methodology to verify that the performance and behavior of TAS agents, once ported into the EAAGLES environment, is consistent with that of the same TAS agents when operating in the JSAF environment. As has already been discussed, we planned to narrow our scope of study to studying the performance of TAS wingmen agents only, in each of three distinct categories:

- General Formation
- Tactical Employment
    - Offensive Employment 2 v 1, 2 v 2
    - Defensive Reactions vs Surface Threats
- Low-Altitude Formation and Employment

Planned Scenario

In order to accomplish each of these three phases, we designed a multi-phase scenario, testing the TAS wingman performance in each of these areas. This scenario was to be flown by each of eight USAF fighter pilots from various backgrounds, each with over 1500 hours of fighter aircraft time. We had hoped that the variations in techniques between pilots of different fighter aircraft types would explore a broader range of maneuvering than the techniques of a single fighter community. The outline that follows is a summary of our verification scenario.

Phase 1: General Formation

Task 1-1: Basic aerobatic formation (simulating close-in tactical maneuvering)

Performance monitored: Formation, Communication (comm.)

Estimated time: 3 Minutes

Overview: This test was designed to mimic a standard “contact” sortie that every military pilot flies during initial pilot training. It was to include one of each of the following maneuvers:

- Loop
- Aileron roll
- Immelmann
- Cloverleaf
- Chandelle
- Split-s

The performance of the wingman would be monitored by:

- Monitoring wingman’s ability to continue in formation
- Monitoring maximum separation between flight lead and wingman through each maneuver using the SAP. Reference Appendix B for additional information about the SAP.
- Monitoring wingman’s comm. should he lose visual and break formation

Task 1-2: Basic Tactical Formation

Performance monitored: Formation, Communication

Estimated time: 5 minutes

Overview: This test examines the wingman's ability to maintain Tactical (line-abreast) formation during standard tactical maneuvering. It was to include (as a minimum) one of each of the following maneuvers:

- 90° Turns Into and Away from the wingman

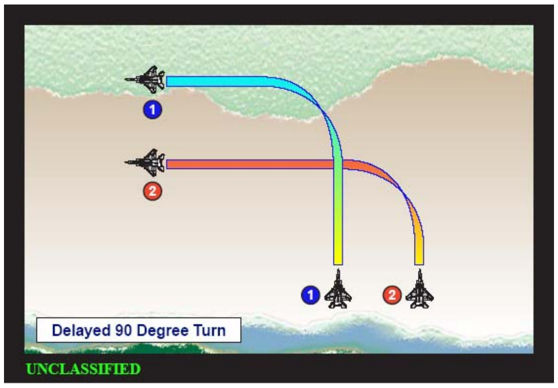


Figure 4. Delayed 90 Degree Turn

- 45° Turns Into and Away from the wingman

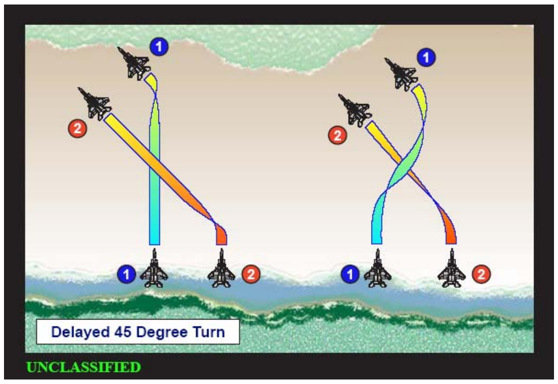


Figure 5. Delayed 45 Degree Turn

- 180° (Hook) turns

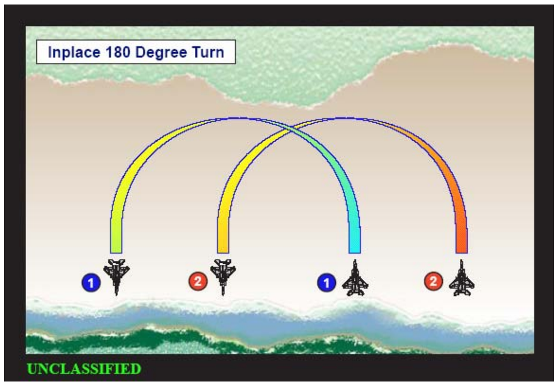


Figure 6. Inplace 180 Degree Turn

- <30° Check turns

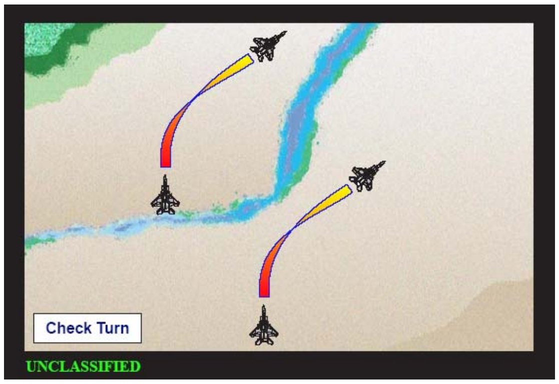


Figure 7. Check Turn

The performance of the wingman would be monitored by:

- Monitoring wingman's ability to continue in formation
- Monitoring maximum separation between flight lead and wingman through each maneuver (using SAP)
- Monitoring the wingman's ability to return to proper tactical formation
- Monitoring wingman's comm. should he lose visual and break formation

Phase 2: Tactical Employment

Task 2-1: 2 v 1 Stern conversion against unaware bandit

Performance monitored: Formation, Tactical Performance, Communication

Estimated time: 4 minutes

Overview: This exercise is a basic station-keeping exercise for a wingman. It was modeled with a TU-95 Bear aircraft at medium altitude. The flight lead runs the intercept, making all the decisions. It would allow us to measure the performance and decision process of the TAS wingman in a benign environment. These runs would be recorded by way of logging the SAP file. This task could also be combined with task 3-2 to save time, though it would be desirable to perform both to observe behavior differences in the two different altitude arenas.

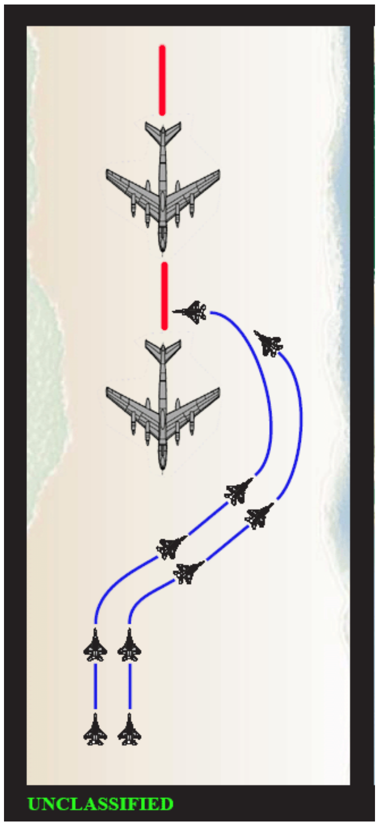


Figure 8. Stern Conversion

The performance of the wingman would be monitored by:

- Monitoring wingman’s formation / station-keeping
- Monitoring wingman’s comm. about target
- Monitoring wingman’s attempts to gain SA on target (radar/visual)
- Observing the wingman’s goal priorities

Task 2-2: 2 v 2 Tactical Intercept

Performance monitored: Formation, Tactical Performance, Communication

Estimated time: 8 minutes

Overview: This task would examine the TAS wingman agent's tactical ability in a moderate threat environment. It was modeled with a 2-ship of MiG-29 aircraft at 25,000 feet, each with a loadout of two AA-10A Alamo (semi-active RADAR) and four AA-11 Archer (heat-seeking) missiles. The MiG formation was tasked for an Offensive Counter Air mission to sweep the area behind the blue aircraft. The TAS wingman would need not only to employ offensive ordnance, but defend against the RADAR threat by performing appropriate flight maneuvers. This is the most tactically challenging of the tasks we designed.

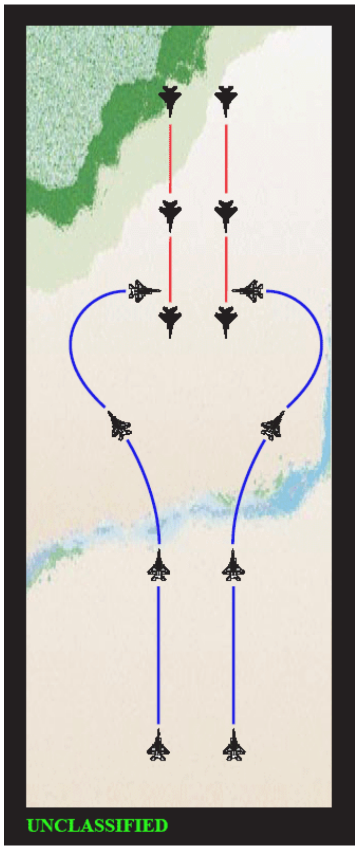


Figure 9. Tactical Intercept

The performance of the wingman would be monitored by:

- Survivability of the wingman
- Monitoring wingman's formation / mutual support
- Monitoring wingman's comm. about target(s)
- Monitoring wingman's attempts to gain SA on target(s) (radar/visual)
- Monitoring wingman's weapons employment mechanics

Task 2-3: Defense against surface threats

Performance monitored: Formation, Tactical Performance, Communication

Estimated time: 2 minutes

Overview: This task measures the TAS wingman's ability to detect, react to, and survive against surface to air threats. This was modeled using an SA-6 Transporter Erector Launcher (TEL) and RADAR with a dedicated comm. link between them. The flight lead intentionally flies into the SAM ring, and then defensively reacts at a predetermined threat range.

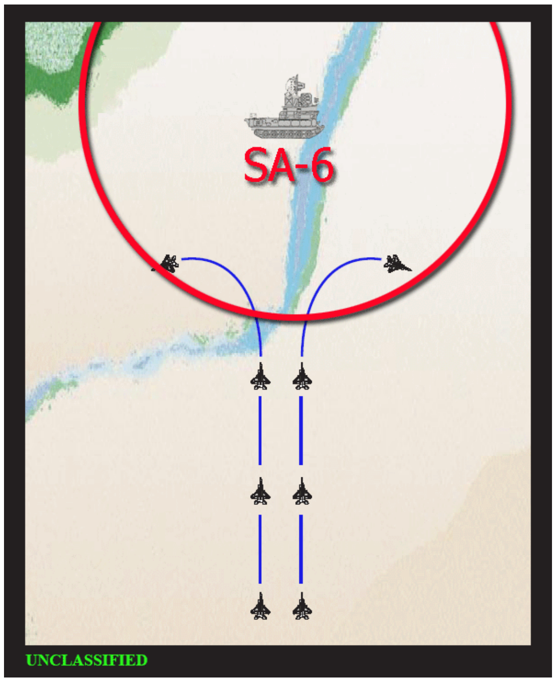


Figure 10. Surface Threat Avoidance

The performance of the wingman would be monitored by:

- Survivability of the wingman
- Monitoring wingman's formation / mutual support
- Monitoring wingman's defensive reactions to the threat

Phase 3: Low-Altitude Formation and Employment

Task 3-1: Basic Formation / Terrain Avoidance

Performance monitored: Formation, terrain avoidance

Estimated time: 3 minutes

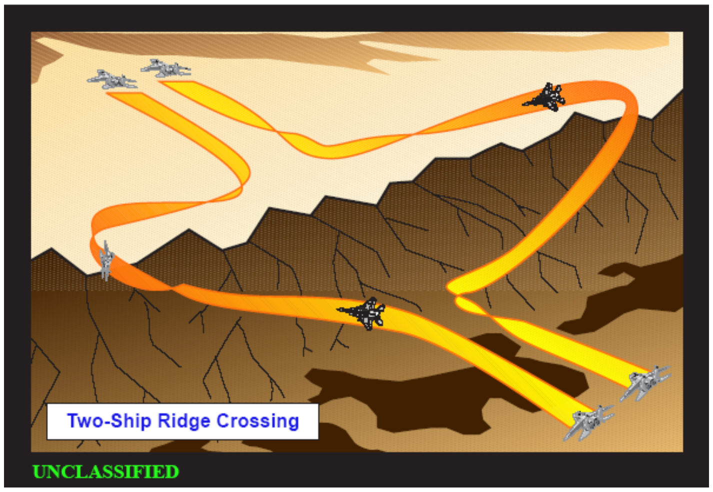


Figure 11. Low Altitude Formation

Overview: This task requires no additional JSAF modeling, but the flight lead must visually navigate to a mountainous area of the terrain, and fly basic terrain masking maneuvers as he would if masking from a RADAR threat. The desire is to observe the TAS wingman's ability to terrain avoid, as well as note the amount of performance degradation, if any, in his maintaining proper formation.

The performance of the wingman would be monitored by:

- Assessing the survivability of the wingman
- Monitoring the wingman's formation / terrain avoidance techniques
- Monitoring the wingman's comm. should he lose visual and break formation

Exercise 3-2: 2 v 1 Low-altitude stern conversion

Performance monitored: Formation, terrain avoidance, tactical performance, Communication.

Estimated Time: 4 minutes

Overview: This exercise is a basic station-keeping exercise for a wingman in the low-altitude environment. It was modeled identically to Task 2-1, but in the low altitude environment. The emphasis here is determining how the wingman's performance changes in the low altitude arena. TAS agent behavior in this task could be directly compared to Task 2-1. If time is short, this task could be combined with Task 2-1.

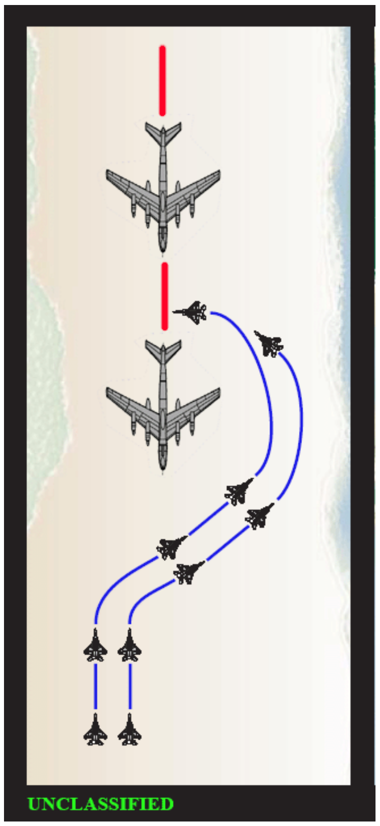


Figure 12. Low Altitude Stern Conversion

The performance of the wingman would be monitored by:

- Monitoring wingman's formation / station-keeping / terrain avoidance
- Monitoring wingman's comm. about target
- Monitoring wingman's attempts to gain SA on target (radar/visual)
- Observing the wingman's goal priorities

Scenario Development Summary

We estimated with setup, rejoin, and familiarization time that the entire scenario could be executed in under 45 minutes. If for some reason either the platform or agent aircraft were to crash or be killed, both systems would require a reset, adding an additional five minutes to the run time.

Scenario Execution Plan

We planned to run this scenario with eight USAF fighter pilots, all very experienced in fighter aircraft, with between 1500 – 2500 fighter hours each. Among the group were to be four F-15 and four F-16 pilots, three USAF Fighter Weapons Instructor Course Graduates, and three Flying Training Unit Instructor Pilots. This test group of pilots is referred to as "platform pilots" throughout the report.

Platform pilots were to be briefed on mission order and parameters, the limitations of the simulator, unclassified nature of the weapons system, and the desired learning objectives for each of the tasks. The platform pilots then were to enter the simulator, while the console operator and the mission results recorder initiated the system as depicted in *Figure 2. Final JSAF / EAAGLES Interface*.

The console operator was to communicate with the EAAGLES platform pilot, and enter any communications to the TAS wingman via the Communications Panel module. He would keep the platform pilot aware of his wingman's position when behind the 3/9 line, as well as offer any targeting or navigation control necessary to effect the tasks. Finally, the console operator was responsible to manually "kill remove" any aircraft that were “hit” by weapons from the EAAGLES aircraft, since JSAF did not perform that task automatically.

The mission results recorder was responsible for ensuring the SAP log files for the TAS wingmen agent were being recorded at the appropriate times. He also was to annotate results of the monitored attributes for each of the tasks. Some of these results were times until events occurred, some were discrete or categorized outcomes. None of the measured characteristics required extremely accurate timing or calculations, so we did not utilize any recording equipment other than the TAS SAP log. This log would provide us with a real-time playback of all recorded sections, from which we could calculate timing if the mission results recorder happened to miss a time.

After the sortie, we planned to debrief the platform pilot, and question him as to what his perceptions were of the TAS wingman’s performance in each individual task and general areas specific to each of the three phases. We planned to use a scale of 1 to 5, 1 being worst and 5 being best. In addition, to evaluate the effectiveness of the DIS “dead reckoning,” questions were planned regarding the fidelity and naturalism of the agent as he appeared from the simulator cockpit. Lastly we wanted to document “perceived workload level” of various tasks, to measure subjectively in relational terms how “tasked” the flight lead felt in leading the element and directing his wingman. These statistics would be recorded and tabulated, and compared to results from the same scenario fully flown in the EAAGLES environment in the comparison phase of the verification.

Scenario Testing

After we had built each task of the three phases of the scenario, we attempted to fly through the scenario ourselves as platform pilots with a TAS agent wingman. It was at this point we made several discoveries of prohibitive limitations in the JSAF/Soar software that caused us to alter our plan. The discoveries are listed by task.

Phase 1: Formation

Task 1-1: Basic Aerobatics

- No significant deficiencies. Results discussed in the next section.

Task 1-2: Tactical Formation:

- No significant deficiencies. Results discussed in the next section.

Phase 2: Tactical Employment

Task 2-1: 2 v 1 Stern conversion against unaware bandit

- No significant deficiencies. Results discussed in the next section.

Task 2-2: 2 v 2 Tactical Intercept

- JSAF 2004 contained code that did not support TAS agents' weapons loading in JSAF. This was not obvious, and was only discovered when we attempted to determine why no TAS agents (flight leads or wingmen) would engage any enemy aircraft, even when ordered to by the controller, and given a hostile declaration. The fix to this error was a simple one, but the resources to obtain the fix were largely unavailable. AFRL Mesa was able to convince SoarTech, Inc. to provide us with the patch. The patch remedied the weapons load problem.

- The SoarSpeak module which had guaranteed the success of such exercises as *Coyote* and *Flight Lead Upgrade* had lost functionality due to numerous upgrades to JSAF code. Because of this, the flight lead was unable to communicate directly with the TAS agent wingman, and relied solely upon the console operator to pass information back and forth to his wingman. In addition, the radio call list in the Communication Panel Module did not contain sufficient commands to effectively order or target the wingman. When monitoring a TAS element (both flight lead and wingman TAS agents), the flight lead would target the wingman using "*Sort azimuth*" or "*Target 360 for 30 miles, 13,000 hostile,*" or other representative AFTTP 3-1 brevity. Without some sort of targeting command, the TAS wingman did nothing more than maintain formation with the flight lead. When we queried Soar Tech, Inc. regarding the possibility of adding more tactical radio calls to the Communications Panel, they informed us that it would be possible but it would require (additional funding and) a programming update to several modules.
- TAS agent wingmen did not react to RADAR threats during any intercept. They elected to stay in formation, and did not show any indication of being aware they were being targeted. If the EAAGLES platform flight lead went defensively to the notch, the TAS agent wingman would not target the threatening aircraft, but remain in formation with his flight lead without any significant tactical action.
- Once the TAS agent wingman identified a bogey/bandit for the first time, he would report it on the primary fighter radio, attempt to keep radar or visual SA, but never act on that object again.

Task 2-3: Defense against surface threats

- No significant deficiencies. Results discussed in next section

Phase 3: Low-altitude Formation and Employment

Task 3-1: Basic Formation / Terrain Avoidance

- We found that the TAS wingmen did not execute terrain avoidance. They would maintain formation with the EAAGLES platform flight lead, and impact terrain if it entered its flight path. Results of our tests are discussed in the next section.
- We found that if the EAAGLES platform flight lead accidentally flew beneath the terrain, causing the TAS wingman to momentarily lose sight, the TAS wingman would assume the flight lead had crashed. If the flight lead subsequently resumed flight above the terrain (crash override turned on in the EAAGLES platform) then the wingman would NOT resume his role as a wingman, and both simulators required a reset to affect another rejoin.

Task 3-2: 2 v 1 Low-altitude stern conversion

- We found the same results at Task 3-1. Behavior was identical to the medium altitude stern conversion of Task 2-1, but any time the formation position brought the wingman's flight path into terrain, the wingman would not terrain avoid, but would impact the terrain.

Due to these findings, our team decided the results of testing our planned eight platform pilots in the scenario would not be meaningful.   We elected instead to run multiple runs on the Tasks that the TAS wingman could perform correctly, and report on those.   At this point we clarify that this effort was intended to be simply a verification that the TAS wingman's behavior would be identical in the EAAGLES environment as it is in the JSAF environment. It was not meant to be a validation of TAS agent behavior. Many studies have been done in the past to verify TAS agent behaviors, though obviously none on this version of JSAF. We were unable to obtain any archived versions of JSAF/TAS in which the TAS agents functioned correctly, as had been reported in *Coyote*, *RoadRunner*, and *Flight Lead Upgrade*. Despite only being a verification, we thought it inappropriate to evaluate and quantify the results of inappropriate behavior on the part of the TAS agents. We will limit the scope of the results to identifying the deficiencies in the current JSAF/TAS model, and continue to expand on the Tasks in which we enjoyed success.

Summary

This chapter provided the details as to how we arrived at a functional system and constructed a scenario for the verification testing of the TAS agents. It first discussed the difficulties involved with finding a combination of working hardware, operating system, JSAF, and TacAir Soar software to build a system that met our needs. This chapter then discussed the relative ease of interface of the JSAF/TacAir Soar software with the software and "human in the loop" hardware in the EAAGLES environment. Lastly, it outlined the details of the planned scenario and testing gameplan that was to be used in the observation phase of the verification methodology, and finally how the results of testing that gameplan created a significant shift in our verification strategy.

IV. Analysis and Results

Chapter Overview

Having significantly altered our strategy, and electing not to pursue benchmarking of the agents using our eight platform volunteer pilots, this chapter discusses the results of the areas in which we enjoyed success in multiple iterative tests of the system. It will outline, by task in the scenario, the ability or inability to evaluate the desired agent function, as well as any conclusions we were able to draw through multiple runs of the scenario. The performance data from each successful task is summarized and evaluated for use in future research.

Testing Results

Phase 1: Formation

Task 1-1: Basic aerobatics

- Each of our team researchers ran fifteen runs of our planned aerobatic profile, recording the SAP log file for playback. Observing the Plan Form Display (PFD), SAP display, and SAP logfile, we determined that the TAS wingman never lost visual, and the greatest separation attained between the flight lead and wingman ranged between .65 and 1.91 NM. The first researcher flew each of the fifteen runs continuously with no pauses between individual maneuvers. After each set of maneuvers, the flight lead gave the Soar wingman a "straight and level," non-maneuvering platform so that he could fully stabilize in his formation position.

Table 3. Maximum Separation From Flight Lead

Run	Max Separation (NM)		Run	Max Separation (NM)	
	Researcher 1	Researcher 2		Researcher 1	Researcher 2
1	1.39	1.00	9	1.74	0.83
2	**1.91**	1.11	10	1.48	1.13
3	1.82	1.18	11	1.48	1.10
4	1.48	0.92	12	1.82	1.07
5	1.48	0.99	13	1.39	1.15
6	1.48	**0.65**	14	1.82	0.97
7	1.56	1.02	15	1.56	1.19
8	1.74	0.95	**Mean**	**1.62**	**1.01**
			Variance	**0.0322**	**0.0209**

- Researcher #2 then accomplished 15 runs of the aerobatic profile as the flight lead. The only difference in this set of runs was that the flight lead allowed the Soar wingman to correct his formation position between each maneuver, not just between runs. *Figure 14* depicts the separation between researcher #2 as the flight lead and the TAS wingman during the first run of the acrobatic profile. The jagged lines demonstrate the oscillatory reactions of the wingman in his attempt to maintain position.

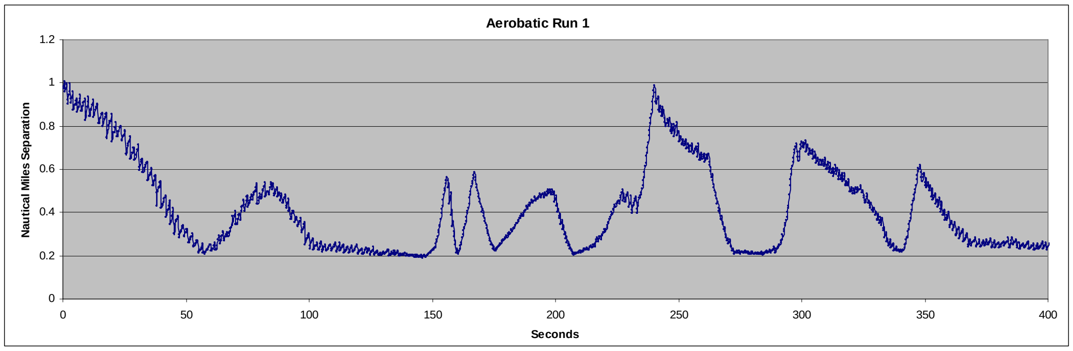


Figure 13. Distance from Flight Lead during Aerobatics: Run 1

- In general, the wingman's performance in maintaining formation was good. The goal stack in SAP indicated the wingman maintained good situational awareness (SA) of the flight lead. As the maneuvering increased, the wingman's top priority was to maintain formation, adjusting all flight characteristics to maintain the position of combat spread (line abreast). However, anytime the flight lead made aggressive changes in direction using vertical maneuvering, the Soar wingman almost exclusively maneuvered in the horizontal to change his flight path and follow lead. This resulted in the wingman being "spit out" of formation during maneuvers in which a human wingman would easily be able to maintain formation. This was primarily noted during maneuvers which involved simultaneous, rapid changes in both airspeed and altitude (chandelle, cloverleaf).

- We also noted that, through maneuvering geometry, the Soar wingman's flight path passed far too close to his flight lead's aircraft. The minimum range noted was 13 feet. Normally, during this type of maneuvering, it is accepted that the range between aircraft may collapse to as little as 300 feet. The Soar wingman flew inside this range on five of the 15 runs with researcher #2. The Soar wingman was not noted to have flown within 300 feet of the flight lead with researcher #1 due to the manner in which the profile was flown.
- The results of the maximum separation measurements between the profiles of researcher #1 and researcher #2 demonstrated that differences in flying and profile management styles resulted in two statistically different outcomes. We performed a two-sample t-test on the two sets of data which rejected the null hypothesis that the samples shared the same mean. Figure 14 shows a histogram of the maximum separation of the wingman during each profile run. Figure 15

demonstrates the difference between the sample distributions of the two researcher's results, and shows that they are statistically significantly different.

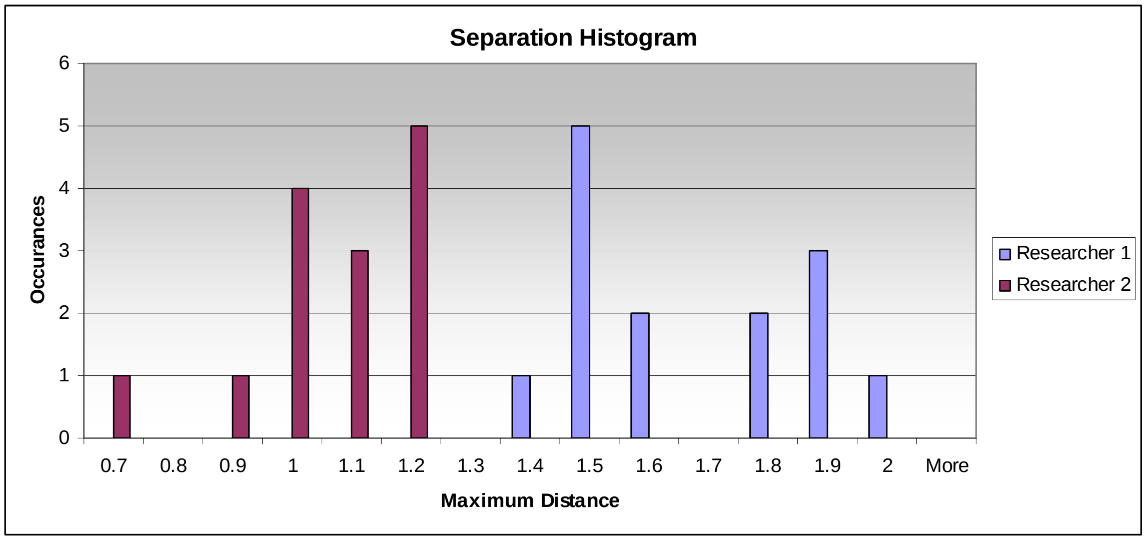


Figure 14. Maximum Separation Histogram

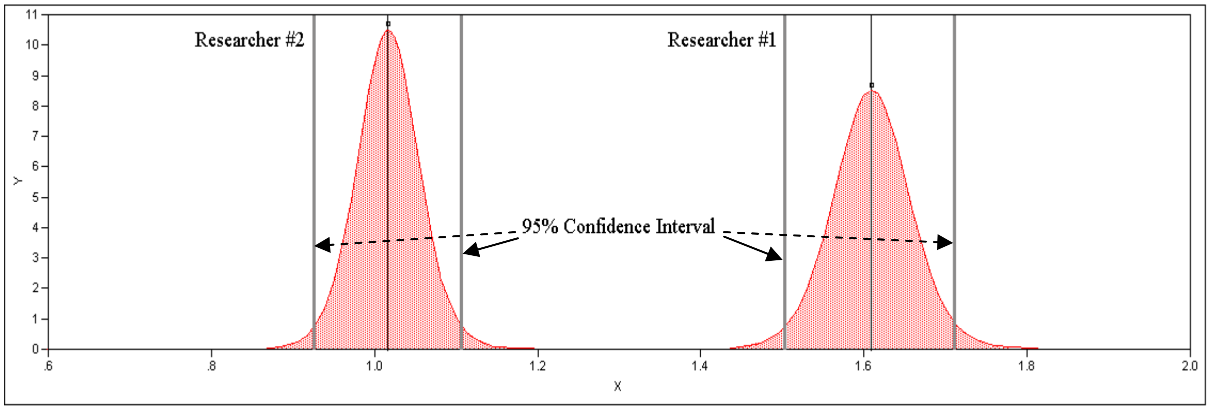


Figure 15. Difference in Separation Distance

- We intentionally flew the profiles with only the information we had planned to brief the platform pilots, with no other discussion of techniques between us. This demonstrated that, in a verification methodology flown with multiple platform pilots, either extremely specific instruction on how to fly and manage the profile must be given, or a relatively large number of platform pilots must be observed for their mean results to be considered. Given the constraints of available platform pilots and simulator resources, more stringently defining the profile is preferred.

Task 1-2: Tactical Formation:

- Like the basic aerobatics, we ran fifteen runs of tactical formation, and recorded the SAP logfile for playback. Observing the PFD and the SAP display, we determined that the TAS wingman never lost visual, and the greatest separation attained between the flight lead and wingman ranged from 1.0 to 1.2 NM.
- We found characteristically the TAS wingman made aggressive corrections in direction and altitude, and speed-matching, but not position. Standard military formations usually involve the wingman maintaining formation on or behind the flight lead's 3/9 line without being directly behind him, as depicted in *Figure 3. General Formation Zones*. In an effort to maintain good fighting airspeed at all times, wingmen are taught to correct their position using maneuvering geometry, not large changes of airspeed. The TAS wingman tended to accept being directly in front or behind the flight lead (out of position), and fixed his 3/9 line position exclusively with airspeed adjustments

Phase 2: Tactical Employment

Task 2-1: 2 v 1 Stern conversion against unaware bandit

- After test runs as discussed in the last section, the TAS agent wingman did not perform any tactical operations after the initial radio call to identify the bandit. In our initial assessment runs, each time the wingman maintained formation while gaining RADAR SA on the bandit. When within visual range parameters of the bandit, the wingman attained a tally (visual contact, as verified on the SAP) and continued in formation as the flight lead ran the intercept. No additional action was noted. We therefore did not perform any additional test runs as the results would have no significance.

Task 2-2: 2 v 2 Tactical Intercept

- As noted in the previous section, no additional runs were performed after our initial assessment runs. The wingmen could not be targeted into the group by the flight lead, and therefore simply maintained formation throughout the intercept. Deficiencies were discussed in the previous section.

Task 2-3: Defense against surface threats

- We ran 15 runs of the defined scenario against the SA-6. In each case, the EAAGLES platform pilot flew directly at the SA-6 threat inside the threat ring. Once there was indication of SAM launch, or the flight lead reached a predetermined distance into the threat ring, the flight lead dragged directly out of the circle. In all fifteen test runs, this resulted in the same approximate flight path for the flight lead in the SAM ring, but different paths for the wingman:

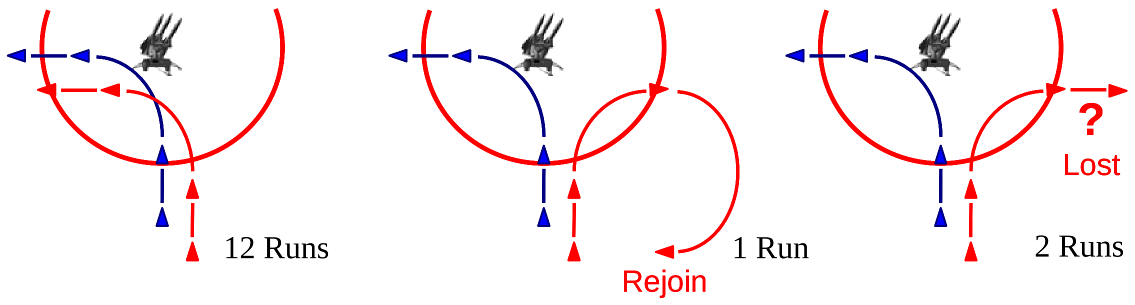


Figure 16. Resultant Aircraft Flight Paths

- In all fifteen runs, the wingman, immediately upon entering the threat ring, displayed SA of the SAM's RADAR status, and maneuvered defensively to exit the threat ring. In twelve of the runs, the TAS agent turned with the flight lead, in three it turned away, placing the threat ring between himself and the flight lead. In all the cases the wingman attempted to maintain visual contact with the flight lead.
- In the three cases that the TAS agent turned away, the wingman became geographically split from the flight lead. The agent prioritized staying out of the SAM threat ring over maintaining formation. In two of the three runs, the TAS agent lost visual contact with the flight lead and began operating independently. In the third run, the wingman was able to maneuver around the ring and rejoined with the EAAGLES platform flight lead.

Phase 3: Low-altitude Formation and Employment

Task 3-1: Basic Formation / Terrain Avoidance

- We ran 15 runs through various areas of terrain in the Southwest US (SWUS) terrain database. In each run, the flight lead maintained between 200' and 500' AGL, performing standard ridge crossing and terrain masking techniques.

Because the exact flight path through the terrain varied in each run, we recorded only whether the wingman impacted terrain or not. The wingman impacted the terrain in under 90 seconds of low-altitude operation in all 15 runs.

- The behavior and priorities of the wingman appeared to not change from its behavior and priorities at medium altitude. His formation characteristics appeared to continue with no apparent concern for the terrain. Nowhere in the goal stack did any goal of "avoid terrain" ever appear. We saw no evidence that the wingman was even aware of the terrain throughout all 15 trials.

Task 3-2: 2 v 1 Low-altitude stern conversion

- Due to the deficiencies noted in Task 3-1, no additional runs were performed or data collected in this area.

Summary

This chapter outlined the performance of the TAS agents by each task they were to perform in the scenario. Due to lack of functionality in several areas of JSAF and the TAS agents, the decision was made not to benchmark the agents using the planned eight platform pilots; therefore, we were not able to evaluate all the agents' task skills necessary for a complete comparison in the EAAGLES environment. There were several task areas in which we enjoyed success, and this chapter outlined the results of the testing in those areas.

V. Conclusions and Recommendations

Chapter Overview

This chapter discusses the outcomes and conclusions of our research, encompassing scenario development, testing, and execution. Having drastically altered our gameplan, the results were limited to the tasks that we found functional in the current JSAF/TacAir Soar combination. This chapter will delineate our conclusions and recommended decision, as well as the primary findings that support that decision. It also presents an exhaustive discussion on the lessons that were learned regarding the various components of the system, their interaction, and the agencies that created, use, support, and require them. Lastly it describes some recommendations for future research in the effort to measure the validity of TacAir Soar agents in the EAAGLES environment.

Conclusions of Research

In deriving a methodology to validate the performance of TAS agents in the EAAGLES environment, we faced several challenges. First, the software to incorporate the agents in EAAGLES has not yet been completed. Second, the version of the agents to be incorporated remains under construction. Lastly, the most recent recorded valid performance of TAS agents is over 7 years old, spanning several software version changes in each of the many modules: Linux, JSAF, HLA RTI's, and TacAir Soar. With the first two limitations, we elected to design a methodology to verify that the performance of the TAS agents did not change between their operations in JSAF and those in EAAGLES. We would not discover the third limitation until well into our development process. Because of the large number of roles and missions a TAS agent can simulate, we narrowed our methodology to a single mission and role: a wingman agent in a DCA role. Once the methodology was established, further scenario modifications could be made to verify the remaining roles and missions.

We developed a scenario to test the wingman agent in each of three distinct categories:

- General Formation
- Tactical Employment
    - Offensive Employment: 2 v 1, 2 v 2
    - Defensive Reactions vs Surface Threats
- Low-Altitude Formation and Employment

The scenario was planned in detail encompassing subject selection, task selection, briefing, execution, and debriefing, as well as data collection. The results would provide a statistical basis for comparison of the agents in each of the two simulation environments.

The research for this project revealed several difficulties in using JSAF. First, the initial investment in achieving an operational system meeting the requirements of this effort was much greater than anticipated. Second, we revealed that the once-validated performance of TAS agents in JSAF has decayed as the JSAF environment and software has changed to emphasize surface warfare. Finally, we were able to make recommendations for validation methodology based upon the findings of our research.

Attaining, installing, and repairing the Linux/JSAF/TAS combination of software consumed the bulk of our research time, reducing our resources available to better test the system. Through systematic iterations of installing three different versions of JSAF on five different versions of the Linux operating system (OS), we were able to achieve a system marriage that met our requirements for research. To reduce the time spent in this avaricious process, we have outlined specific installation instructions. Appendix A enumerates specific instructions for Linux OS, JSAF, and TacAir Soar agent installation. It includes all the necessary tweaks and software patches to facilitate an operational system. Appendix B articulates the process of constructing working TAS scenarios within JSAF, and interfacing with hardware in the EAAGLES environment.

The research in this project revealed a significant degradation in the performance of the JSAF/TAS combination. As is obvious from the results in the previous section, we were only able to verify that the formation and surface-to-air defense logic functioned properly in the current JSAF/TAS package. When Soar Tech, Inc. and JFCOM/J9 JSAF specialists were queried, they both confirmed that the many upgrades to JSAF may have caused errors in the TAS/JSAF interface that have not been found up to this point. It is obvious that no validation has been performed on the TAS agents in the past several renditions of JSAF.

It is our determination that, instead of verifying that the behavior of the TAS agents ported into EAAGLES is equitable to their current performance in JSAF, a more appropriate study would be to perform a full validation of TAS agent behavior in EAAGLES. This conclusion is based on the following findings:

1. JSAF has become disassociated with the TAS agent software during the last several upgrades. The current version of JSAF demonstrates corruption of the

JSAF/TAS/HLA interface. Behaviors documented as functional in exercises *Coyote, RoadRunner,* and *Flight Lead Upgrade* no longer function correctly in the current model. It is impossible to determine the true performance of the agents with the current questionable interface.

2. The versions of software that contained a functional interface are not currently available through JFCOM, Naval Warfare Center, AFRL, or Soar Tech, Inc.
3. Major software upgrades by JSAF developers (at expense) would be required to return the current JSAF/TAS package back to its previously functional state.

The Defense Modeling and Simulation Office (DMSO) provides guidelines as to the verification of simulation systems, and we recommend a baseline scenario be organized for each of the areas of tactical performance that require examination. The verification, validation, and accreditation (VV&A) guide offered by DMSO can be found at http://vva.dmso.mil/Default.htm. Additionally, the *Recommended Practices Guide*, also located at this website, is a rich resource for VV&A across the spectrum of modeling and simulation.

Lessons Learned

1. Linux/JSAF Setup

The JSAF software is very large, consisting of more than 850 individual object libraries. A complex web of dependencies connects these libraries to one another. The JSAF software is written in ANSI C, but has a complex object-based architecture that simulates inheritance and aggregation relationships among libraries. Extensive configuration script and reader (i.e., data) files further add to this complexity (Trevisani, p. 9).

As was noted in Chapter 2, to build and maintain a working Linux / JSAF system requires considerable experience in both pieces of software. To design and run large scenarios in JSAF requires a large, trained and experienced technical staff to control the intricacies of JSAF. There is little to no technical support available, other than by seeking the aid of other users. We found no established users' groups, networks, or repositories of databases or files.

In addition, we found an extreme lack of documentation for JSAF installation and maintenance. Since the JSAF software package is modular and extremely large, and many of the modules further comprise script and data files, many attempts may be required before a successful installation is achieved, especially if the code must be recompiled. Simple changes to a single model can involve modifying dozens of components, C program files, and scripts. Once running, there is limited documentation on the operations of JSAF. The JFCOM JSAF User's guide does not include any information on how to run TAS agents. We researched a company named "BMH Associates" that conducts $2,500, 4-day training courses for JSAF operators and developers. Their documentation is reported to have a chapter on TAS operations. This course would be invaluable to those pursuing future research involving the JSAF/TAS combination. BMH Associates' contact information is listed in Appendix D.

2. TAS/JSAF integration support

Our research revealed that JFCOM's current emphasis in JSAF is developing SOF entities using the Soar interpretive language. Their funding is limited, and therefore they are not currently willing to expend further resources into the development or repair of TAS agent models in JSAF. Although representatives from Soar Tech, Inc. regularly interact with JFCOM, and questions may be fielded about TAS agents during those visits, we were not able to obtain anything other than cursory support via that avenue. Likewise, Soar Tech, Inc. was willing to host our team for a short visit to give an overview of their agents in JSAF, but was unwilling to demonstrate any models or versions of JSAF other than what we brought with us from JFCOM (JSAF 2004). When that version failed to function correctly, they could offer us no other solutions in terms of support without additional contracting and funding.

3. Terrain Database Files

JSAF uses the Compact Terrain Data Base (CTDB) format. This format is specific to ModSAF and its various derivatives (of which JSAF is one). CTDB terrain databases are generally created specifically to support large simulation exercises, therefore the areas of the world for which CTDB databases are available are very limited. CTDB databases are oriented toward supporting ground combat, and therefore tend to contain a relatively high level of detail. However, they also tend to be relatively small in terms of the area covered (Trevisani, p.10). Terrain database files are specific to the version of JSAF used; JSAF 2004 uses CTDB version 8.7 files. JSAF is not backwards-compatible with older CTDB formats.

The US Army Topographic Engineering Center (USATEC) maintains a collection of CTDB databases. Their website repository for databases is located at `http://mel.dmso.mil`. At that website a user can browse current available terrain files, or request files in different versions if the version that is required is not available. USATEC was extremely helpful in providing us (within 24 hours) the *version 8.7* file JSAF 2004 required to operate concurrently with the EAAGLES simulator.
We have provided the standard JSAF 2004 terrain files, as well as the specialized file required to run the JSAF/EAAGLES scenario. Any other files will need to be obtained from USATEC. Further contact information for USATEC can be found at Appendix D.

4. TAS performance in current JSAF

Although TAS agents are advertised as having a large, 9,000 rule 'vocabulary', we found that, during implementation in the current version of JSAF, many of these critical rules and processes are not invoked. The tactical ability of the agent as a wingman was poor, and its mission-task management skills, to include terrain avoidance, were almost non-existent. A critical component of TAS, SoarSpeak, was unusable in current versions of JSAF. This was a great detriment to the project as many of the key wingman-flight lead interactions can only be controlled through this module. It is unfortunate that this module has fallen by the wayside during JSAF upgrades. As mentioned earlier, SoarSpeak was vital to TacAir Soar's success at the *Coyote*, *RoadRunner*, and *Flight Lead Upgrade* exercises.

Due to multiple versions of RTIs, it proved difficult to achieve successful networking in HLA with JSAF and TacAir Soar. This hindrance was primarily caused by the evolution of JSAF versions over the years and multiple builds of each version to satisfy the requirements of different versions of Linux and associated C++ compilers. Although most of the different JSAF machines would recognize and display entities from other machines, most of the tactical interaction between entities was nonexistent (no reactions to or weapons employment on agents from other JSAF nodes). During execution, many errors were logged concerning incompatible TAS agent HLA packet transmissions. JFCOM’s emphasis on SOF agents has driven JSAF’s and TacAir Soar’s HLA composition away from the IEEE standards, as pointed out in some of the “help files” included in the JSAF 2004 installation package.

Appendix A: Instructions for Linux Installation

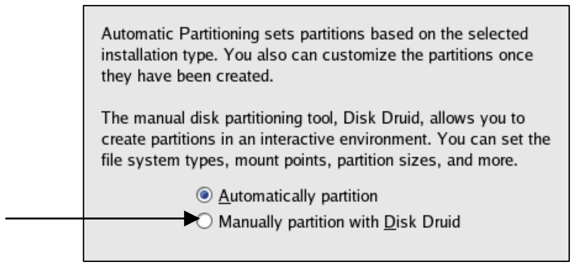
Compatible Linux Installation

If you plan to install Linux on a machine that already has Windows OS, and desire to keep Windows as well as install Linux, you will need to reference the next section, "Dual Booting into Linux."

1. Ensure your computer will boot from the CD drive. You may have to enter the BIOS and select "BOOT FROM CD" prior to moving to the next step.

2. Boot computer to FC3 install disk #1. If you are ONLY installing Linux on the system, continue with these procedures. Once you have initiated the dual boot procedures, return to the next step in this section for installation specifics for Linux.

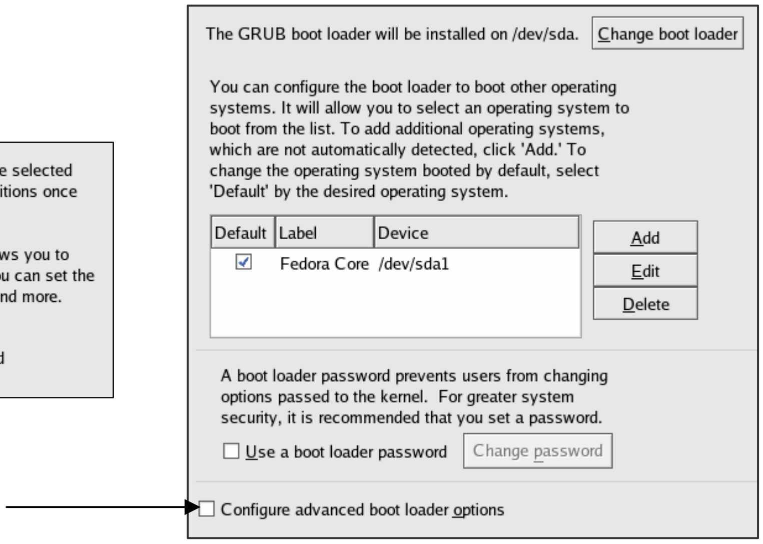
3. Select "custom" for type of installation. Select manual disk partitioning with "Disk Druid." Create a swap partition of at least 1GB (1024MB) and a primary partition. You will have to designate your primary partition as "/" for a successful install. All data on these two partitions will be lost.



Automatic Partitioning sets partitions based on the selected installation type. You also can customize the partitions once they have been created.

The manual disk partitioning tool, Disk Druid, allows you to create partitions in an interactive environment. You can set the file system types, mount points, partition sizes, and more.

(●) Automatically partition
( ) Manually partition with Disk Druid



The GRUB boot loader will be installed on /dev/sda. [Change boot loader]

You can configure the boot loader to boot other operating systems. It will allow you to select an operating system to boot from the list. To add additional operating systems, which are not automatically detected, click 'Add.' To change the operating system booted by default, select 'Default' by the desired operating system.

Default	Label	Device
✓	Fedora Core	/dev/sda1

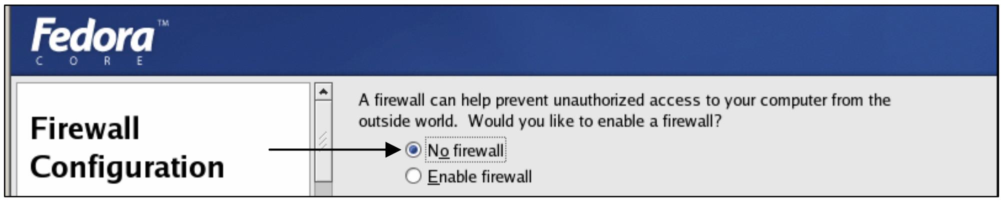
A boot loader password prevents users from changing options passed to the kernel. For greater system security, it is recommended that you set a password.

☐ Use a boot loader password [Change password]

☐ Configure advanced boot loader options

Figure 17. Linux Partitioning Screenshots

Do not enable the firewall, as it will cause great difficulty with the HLA RTI and the DIS gateway. We only recommend you enable this if you have experience with customizing Linux firewalls.



A firewall can help prevent unauthorized access to your computer from the outside world. Would you like to enable a firewall?

[✓ No firewall] Enable firewall

Figure 18. Linux Firewall Configuration

4. Select your installation options. You can select a minimal install, but be sure to select all “Development Tools and “Legacy Support Tools.” Most all of the “Server Tools” are unnecessary to run JSAF, but if you have a need for them, you may select them at this time. We kept the installed items to a minimum as we found it difficult to determine what additional Linux packages slowed JSAF’s performance. Your final install size will be as low as 1750MB.

5. When you are asked for “root” password, realize that “root” is the Linux version of “System Administrator.” Be sure to remember this password as you will need to be logged in as “root” to perform most of the installation options required in this guide.

6. Once the installation is complete, you will be asked to reboot. Don’t forget to remove the installation CD and/or reset the BIOS to “Boot from Hard Disk.” After reboot, allow the Linux OS to load. At this time you may create an additional user (if desired) so that you don’t always have to be logged in as “root.” Enter the locale, date, time, and screen resolution specifics, and log into Linux for the first time as “root,” using “root” as the username, and the designated password.

Dual Booting into Linux

These instructions apply if:

- Your machine already has Windows installed, and you are installing Linux as a second operating system, and
- You want to leave the Windows boot loader (NTLDR) on the MBR (Master Boot Record). This allows you to continue to boot Windows with no issues. Windows 2000/Windows XP or anti-virus software may cause errors if the MBR does not contain the Windows boot loader

Requirements for `/boot` Partition

The location of the `/boot` partition on the hard drive is critical so that the BIOS 1024 cylinder limit doesn't limit your system. The BIOS of older systems can't access data beyond cylinder 1024, which is ~8.5 GB. A simple way to avoid the BIOS 1024 limit is to create `/boot` within the first 1024 cylinders (~8.5 GB) of the hard drive. **If you have multiple hard drives (disks)**, `/boot` must be on the same hard drive (probably the first hard drive) that has the Windows boot loader (NTLDR) on the MBR.

The /boot partition is simply where the boot record for Linux resides. You may place the remainder of the operating system on a different partition, which means whatever partition contains /boot can be very small.

Here are some options for where to create `/boot` partition.

1. Shrink the Windows partition such that there is 75 MB of unused disk space at the beginning of the drive and lots of space after the Windows partition. You can install the `/boot` Linux partition in this first 75 MB and avoid any potential issues with the 1024-cylinder limit entirely. This is the option we used.
2. Shrink the Windows partition such that it does not cross the 1024 cylinder (~8.5 GB), and install the `/boot` partition right after the Windows partition.
3. Use LBA (Logical Block Addressing). LBA allows you to boot beyond the 1024 cylinder. In order to use LBA, your BIOS must support it. In addition, for LILO, you must also add a flag to enable LBA support. GRUB supports LBA "out-of-the-box"

To non-destructively shrink the Windows partition, you can use the commercial product Partition Magic. It has an easy-to-use GUI. Unfortunately, the tool that comes with Fedora Core 3, Disk Druid, does not have the ability to shrink existing partitions. Once you've shrunk the Windows partition, you can use Disk Druid during the Red Hat Installation to create all the partitions you need for Linux.

Dual-Boot Setup During Linux Installation

Following are the steps to get dual-boot working with GRUB. This procedure works for Windows 2000 and Windows XP, and should work on Windows NT (all 3 OSs use the same booting architecture).

1. Install GRUB on the first sector of whichever partition you created as the `/boot` partition. **DO NOT INSTALL IT ON THE MBR!**.

   If you are performing the Red Hat installation, for the "Boot Loader Installation" screen:

   - Select "Use GRUB as the boot loader"
   - Select Install Boot Loader record on "...First sector of boot partition".
   - See “Installing Linux” section for the remainder of the Fedora Core Linux installation, then return to the next step here.
   - After finishing the Red Hat installation, reboot into Linux. If you don't have a boot disk, try booting in Linux rescue mode (using either the Rescue CD, or Installation CD #1).

2. Once you get to a command prompt, determine which partition contains the `/boot` partition by running the `df` command. You'll see output like this:

```
Filesystem   1k-blocks   Used      Available   Use%  Mounted on
/dev/hda3    8665372     1639580   6585612     20%   /
/dev/hda2    46636       5959      38269       14%   /boot
/dev/hda6    513776      189504    324272      37%   /osshare
none         256624      0         256624      0%    /dev/shm
```

   From this output, we see that */boot* is on */dev/hda2*.

3. Make a copy of the Linux boot sector onto a floppy or onto a FAT32 partition. We'll name this copy *linux.bin*.

   To make a copy onto a floppy, you may have to mount the floppy drive:

   - Ensure a directory exists to mount the drive to. Change directory to /mnt using the command:

     cd /mnt.

Examine the files using the command **ls**. If a “floppy” directory does not exist, create one using the command

```
mkdir floppy
```

This will give you a folder to “mount” your floppy drive to. (Note: you may have to do this for your CD-ROM, USB key, and other removable media devices.)

- Mount the floppy drive if it's not mounted:

```
mount -t msdos /dev/fd0 /mnt/floppy
```

- Run the following command:

```
dd if=/dev/hda2 of=/mnt/floppy/linux.bin bs=512 count=1
```

Substitute the path for the **`if=`** parameter (the **i**nput **f**ile) with the appropriate partition from the previous step. E.g., set **`if=`** to **`/dev/hda2`**.

- Prior to ejecting the disk, or shutting down, make sure you *unmount* any mounted disks, using the command:

```
umount /mnt/floppy
```

or whatever device is appropriate. If you reinsert the disk, you will probably have to remount it.

4. Reboot into Windows

5. Copy the `linux.bin` file to `C:\`

6. Run notepad and edit `C:\boot.ini`. Note that `C:\boot.ini` is a hidden system file, so it probably won't show up in Windows Explorer. To edit the file, try: `Start->Run` and enter: `notepad C:\boot.ini`. Add the following line at the end:

```
c:\linux.bin="Linux"
```

Note that you must edit `C:\boot.ini` as a user with administrator-level privileges.

To make `C:\boot.ini` writable, you can either :

- Use Explorer:

        - Go to `Tools->Folder Options->View` and select `Show hidden files and folders` and deselect `Hide protected operating system files (Recommended)`.
        - Right-click on the file, view the `Properties` and uncheck `Read-only`. You can now edit the file.
        - After editing the file, restore the settings to their original state.
    - Use the command-line:
        - Make the file writable:

```
attrib -R -S -H C:\boot.ini.
```

        - After you've finished editing the file, put the settings back:

```
attrib +R +S +H C:\boot.ini
```

7. Reboot again. You should be able to pick either Windows or Linux. Selecting Linux will start GRUB.

Created with help from: http://www.geocities.com/epark/linux/grub-w2k-HOWTO.html

NTFS Connection

1. Browse to the cd using the command:

   ```
   cd /media/cdrecorder
   ```

2. Install the rpm with the command:

   ```
   rpm -ihv kernel-module-ntfs-2.6.9-1.667-2.1.20-0.rr.3.3.i686.rpm
   ```

   ```
   Preparing...      ################################# [100%]
     1:kernel-ntfs   ################################# [100%]
   ```

   There should be no errors, just #'s. This is the only command we actually needed, but we'll go on and test what we have done.

3. Next load the kernel module with the command:

   ```
   /sbin/modprobe ntfs
   ```

   There should be no output.

4. Next you will mount the drive, but first you need to know which device your NTFS Volume is on and you will need to create a directory as a mount point. Execute the command:

   ```
   /sbin/fdisk –l
   ```

   The output might look like:

   ```
   Disk /dev/hda: 64 heads, 63 sectors, 4465 cylinders
      Units = cylinders of 4032 * 512 bytes
        Device      Boot   Start   End       Blocks      Id     System
        /dev/hda1          1       2125      4283968+    07     NTFS/HPFS
        /dev/hda2          2126    19851     35735616    0f     Win95  (LBA)
        /dev/hda5    *     2126    4209      4201312+    83     Linux
       /dev/hda6           4210    4465      516064+     82     Linux swap
   ```

   Find the device containing the windows (NTFS) directory, and type the following three commands, substituting your device wherever you see *hda1*:

   ```
   mkdir /mnt/windows
   ```

```
mount /dev/hda1 /mnt/windows -t ntfs -r -o umask=0222
ls -l /mnt/windows
```

The output should resemble the following, and will be listing the contents of the Windows drive you just mounted:

```
...
-r-xr--r-- 1 root root  9719    Aug 24 1996 ansi.sys
-r-xr--r-- 1 root root 15252    Aug 24 1996 attrib.exe
-r-xr--r-- 1 root root 28096    Aug 24 1996 chkdsk.exe
-r-xr--r-- 1 root root  5175    Aug 24 1996 choice.com
...
```

Hopefully everything is working for you now.

5. You may set up the system so the drive automatically mounts every time you boot by adding a line to */etc/fstab* (filesystem table). Open the File Browser, navigate to */etc*, right click on *fstab* and select "Edit with text editor." Add the following line to the bottom of the file (using the appropriate *hda* device in place of /dev/hda1):

```
/dev/hda1    /mnt/windows          ntfs   ro,umask=0222    0 1
```

Now you can browse to */mnt/windows* and **read only** the files on your windows drive (or other drives).

Java installation

In order to correctly install the Java Resource Environment to use the Situational Awareness Panel within JSAF, you will need a functioning connection to the internet. Fedora Core 3 is very strong with automatically configuring your internet connection. There are several FAQ's online should you be unable to connect immediately.

We have outlined the JAVA installation method using Dag Wieers's (http://dag.wieers.com/packages/) Java Runtime RPMs:

1. From the JSAF_CD, copy the *yum.conf* file into the /etc directory using the command:

```
cp /media/cdrecorder/yum.conf  /etc/
```

2. To ensure you can establish a "secure" connection to the source files, enter the command:

```
rpm --import http://dag.wieers.com/packages/RPM-GPG-KEY.dag.txt
```

3. Install the Java Resource Environment from this site using the command:

   ```
   yum --enablerepo=dag install j2re mozilla-j2re
   ```

   This will also install the browser plugin for java.

Java is now installed, and should require no further work from you.

If you are more proficient with Linux, we have provided the necessary files on the CD for manual setup. Reference Appendix C for CD file listing.

JSAF_2004 Installation

1. Open a terminal window and browse to the /usr directory using the command:

   cd /usr

2. Make a directory called “stow” (synthetic theater of war) using the command:

   mkdir stow

3. Make the directory executable by all users using the command:

   chmod 777 stow

4. Copy the “JSAF_CD” folder into the /usr/stow directory using File Browser, or in the terminal using the command:

   cp /media/cdrecorder/JSAF_CD /usr/stow.

   If this gives you an error file, your CD may be called something else. Browse to **cd /media** and **ls** to see what the name of your CD is. Replace **cdrecorder** in the above command with whatever your CD Rom is named.

5. Browse into the newly copied JSAF_CD folder using the command:

   cd JSAF_CD

6. Execute the batch file to install the executables for JSAF_2004 using the command:

   ./ExecutableInstall.sh /usr/stow

   If for some reason you desire to install in a different location, change /usr/stow here to that location. Realize that in several of the batch files, JFCOM has hard-coded this path, so moving JSAF to a different path will possibly make you edit other script files.

7. Back up one level in the file structure using **cd ..** and look at the files in the /usr/stow directory using the command **ls**. You should see:

   ```
   build    JSAF_CD       JSAF_SAVES  terrain
   ```

Each of these new items are folders created by the batch file that you just ran.

8. Create a directory for your exercises using the command:

    mkdir exercises

9. Now you must apply a patch written by Soar Tech, Inc. to fix a bug in the agents' weapons handling. Type the command (on one line):

    cp /media/cdrecorder/patches.soar usr/stow/build/JSAF/data/libsafsoar/agent/

    You will need to answer *yes* when it asks you if you want to overwrite the previously existing file.

10. Now exchange the JSAF CD in the drive for the TERRAIN CD. Browse to the CD using the command:

    cd /media/cdrecorder

    Browse into that, then into the Terrain_CD folder using:

    cd Terrain_CD

11. Install the terrain files using the command:

    ./TerrainInstall.sh /usr/stow

    This will uncompress the terrain files and place them in the /usr/stow/terrain folder. The following terrain sets are installed at this time:

    a. *world_thin_fmt8_7_180W180E75S75N_v01* – This is a low-detail database of terrain covering the entire globe. If, in JSAF, you leave the *detailed* area of your exercise terrain file, JSAF will default to using this data so your entities don't fall off the edge of the earth.

    b. *jak_juo_041101_v09_ctdl_fmt8_7* – This is a high-level database of the JUO federation normal area of operation in Africa. This is the default if you run JSAF without specifying a different terrain database.

    c. *swus_tsga_6x6_v4_020812_ctdl_fmt8_7* – This is a high-level database of southern California and western Nevada. This matches closely to the database AFRL uses for their EAAGLES simulator, and will allow entities in both environments to utilize the same terrain data.

These databases have all been formatted to CTDL version 8.7 specifically for this version of JSAF. We discuss CTDL more in the *Lessons Learned* section of this paper. Go there to find out how to obtain terrain databases for other parts of the globe.

Appendix B: Instructions to run TacAir Soar Agents

Introduction

This guide is not an inclusive JSAF guide, but is intended to give the user enough guidance to design a scenario, create and assign agents, and successfully load them all in JSAF 2004.

Creating a scenario containing TacAir Soar agents

To use TacAir Soar agents in JSAF 2004, they must be created outside of JSAF using an external program called "Exercise Editor." This program may be accessed by creating a terminal window, and browsing to the *runscripts* folder using the following command:

```
cd /usr/stow/build/JSAF/runscripts
```

The Exercise Editor can then be run using the following syntax (all on one line):

```
./ExerEdit –ex blue1 –ex_path /usr/stow
        –terrain swus_tsga_6x6_v4_020812.dir –federation juo
```

The parameters for this script are:

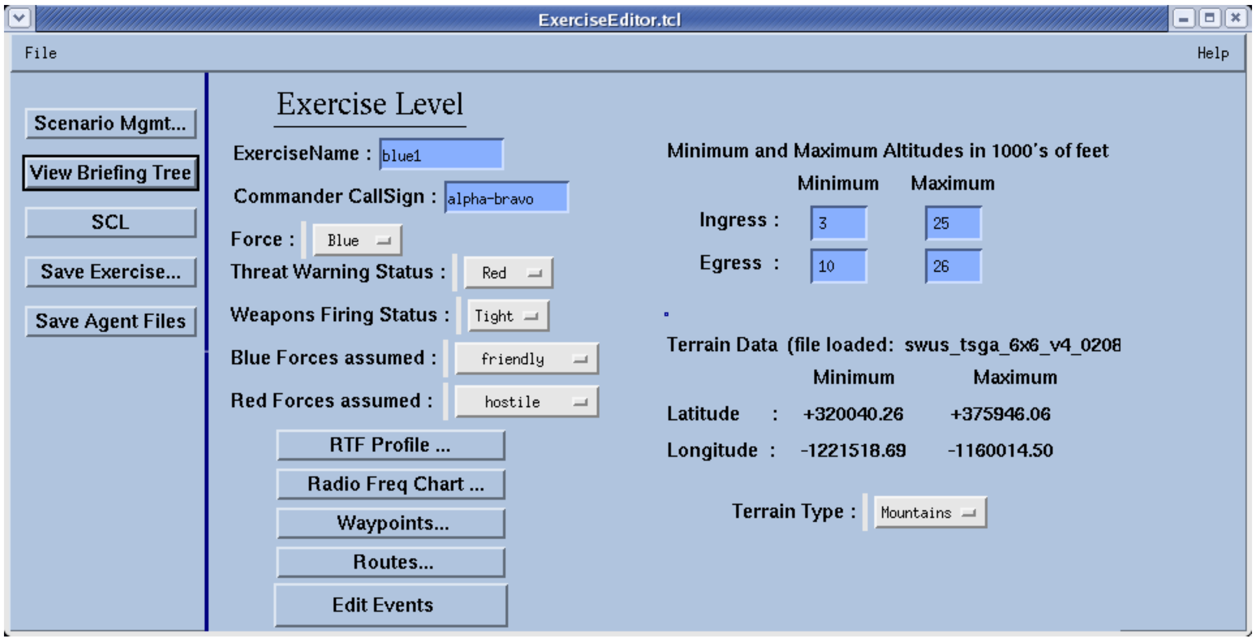
-ex <exercise name> Whatever you want to call your exercise. If you name an exercise that has not yet been created, the script will prompt you and create the exercise folder for you at this time.

-ex_path <path_name> The path that contains your *exercises* folder you created in setup. If you followed the instructions in this paper, that value will be */usr/stow*.

-terrain <terrain folder> The terrain file you plan to use with this exercise. The lateral boundaries of your exercise are defined by the boundaries of your terrain file. For working with AFRL's simulator, we always used the *swus* terrain file listed above.

-federation <federation> There are several federations available, *juo* is the one that works best for stable HLA communication at the facilities utilized for this paper. Note that *juo* is the default if you do not include this option when you run the script.

Once you have run this, the Exercise Editor will open its GUI interface.



ExerciseEditor.tcl

File   Help

Exercise Level

ExerciseName :	blue1
Commander CallSign :	alpha-bravo
Force :	Blue
Threat Warning Status :	Red
Weapons Firing Status :	Tight
Blue Forces assumed :	friendly
Red Forces assumed :	hostile

Minimum and Maximum Altitudes in 1000's of feet

	Minimum	Maximum
Ingress :	3	25
Egress :	10	26

Terrain Data (file loaded: swus_tsga_6x6_v4_0208

	Minimum	Maximum
Latitude :	+320040.26	+375946.06
Longitude :	-1221518.69	-1160014.50

Terrain Type : Mountains

Figure 19. Exercise Editor

TacAir Soar agents are defined in the Exercise structure, with four levels: Exercise, Event, Mission, and Element. To ther right is an example of the Briefing Status Tree view of a simple Exercise scenario containing a single element of F-15C aircraft. Each level of the exercise must be defined appropriately in order to generate TacAir Soar agents correctly. From any screen, you can select *View Briefing Tree* to get this screen, and click on the desired location to navigate to.

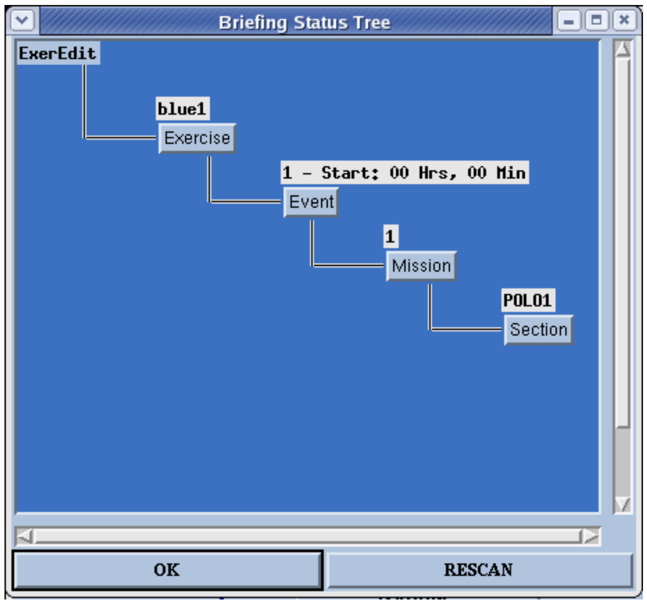


Figure 20. Briefing Status Tree

Exercise Definition

Each exercise may only contain the forces for a single force, (red, blue, brown, etc.). If you wish to create TAS agents for the red force in addition to the blue force, you will have to create two separate exercise scenarios. The main screen allows you to select your defined force, weapons firing ROE, who is hostile to your force, and the type of terrain to prepare your agents for (mountainous, ocean, etc.). Waypoints and Radio Frequencies must be designated from this page before the next levels of the structure can be created.

Waypoints

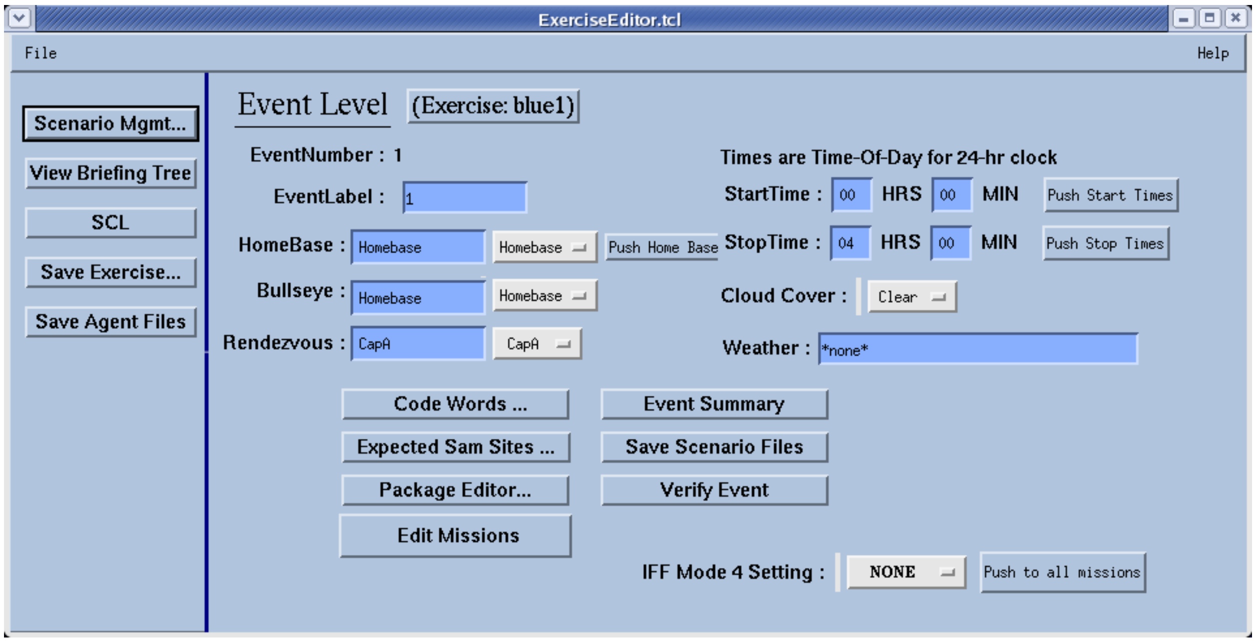
From the initial Exercise Editor screen, you will need to create (at a minimum) a few waypoints. Click on the *Waypoints* button, and enter your desired waypoint locations. There is no GUI to find the coordinates; you must know the location of your desired points in Lat/Long format. You will need, at a minimum, to enter at least one *Other* point to designate the airfield from which the TAS agents take off, and at least one *CAP* point to define the aircraft's cap.

Radio Frequencies

To set up the communication plan for the TAS agents, select the *Radio Freq Chart* button, and enter frequencies for the various radios in the comm. plan. This is necessary to establish communications between the aircraft in the element, as well as between different elements (you may choose to add an AWACS to the exercise, for example).

Once you have completed your selection, select *Edit Events* to create an event. You will return to this screen at the end to save your final exercise scenario.

Event Level



Event Level (Exercise: blue1)

EventNumber : 1

EventLabel : 1

HomeBase : Homebase Homebase

Bullseye : Homebase Homebase

Rendezvous : CapA CapA

Times are Time-Of-Day for 24-hr clock

StartTime : 00 HRS 00 MIN

StopTime : 04 HRS 00 MIN

Cloud Cover : Clear

Weather : *none*

IFF Mode 4 Setting : NONE

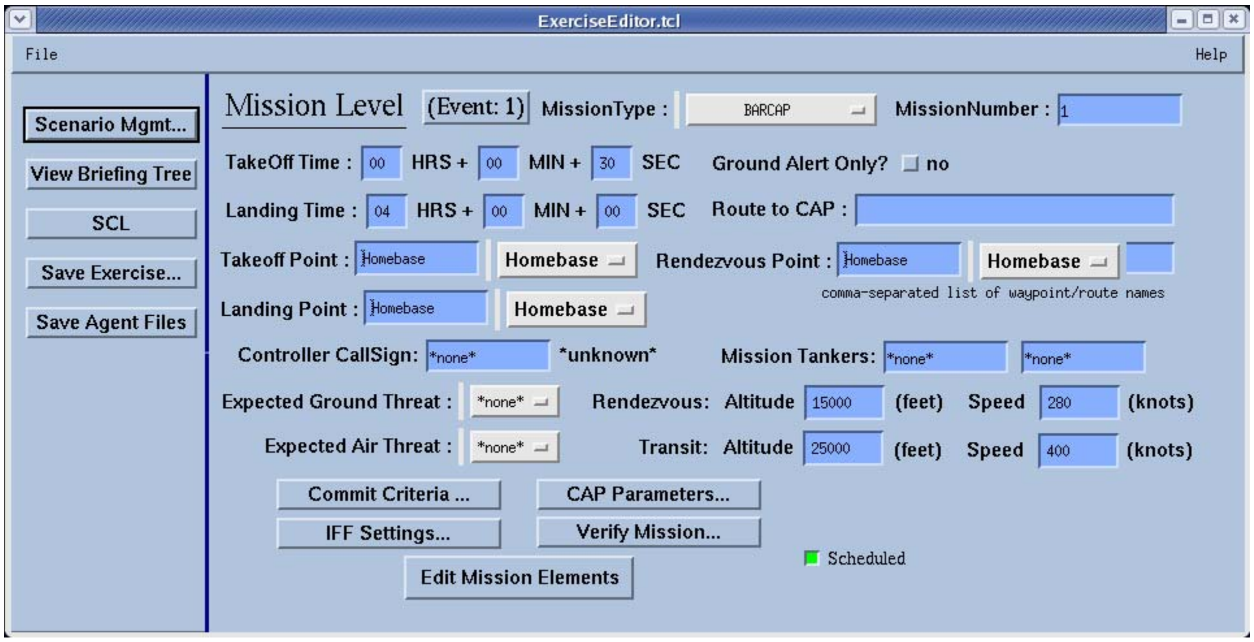
Figure 21. Exercise Editor - Event Level

The event can be thought of as the "package" of aircraft, all the missions for aircraft with this package will be created subordinate to this level.

On this screen you must define the event duration (start and stop times), Homebase, and bullseye points. You may also designate weather, IFF codes, and define SCL loadouts. It is not necessary to define the SCL here, as each aircraft mission element can have its own custom loadout. The *Verify Event* button is used once everything in the sublevels has been defined to check for errors or omissions. You may (optionally) enter expected surface threats, so the agents can "mission plan" their tactics against them. Note that you can navigate back to the *Exercise* level by clicking on the *(Exercise blue1)* button at screen top.

Once this information has been defined, click on the *Edit Missions* button. You will return to this page to save the scenario files when the missions and elements have been defined.

Mission Editor



Mission Level (Event: 1) MissionType : BARCAP MissionNumber : 1

TakeOff Time : 00 HRS + 00 MIN + 30 SEC Ground Alert Only? no

Landing Time : 04 HRS + 00 MIN + 00 SEC Route to CAP :

Takeoff Point : Homebase Homebase Rendezvous Point : Homebase Homebase

comma-separated list of waypoint/route names

Landing Point : Homebase Homebase

Controller CallSign: *none* *unknown* Mission Tankers: *none* *none*

Expected Ground Threat : *none* Rendezvous: Altitude 15000 (feet) Speed 280 (knots)

Expected Air Threat : *none* Transit: Altitude 25000 (feet) Speed 400 (knots)

🟩 Scheduled

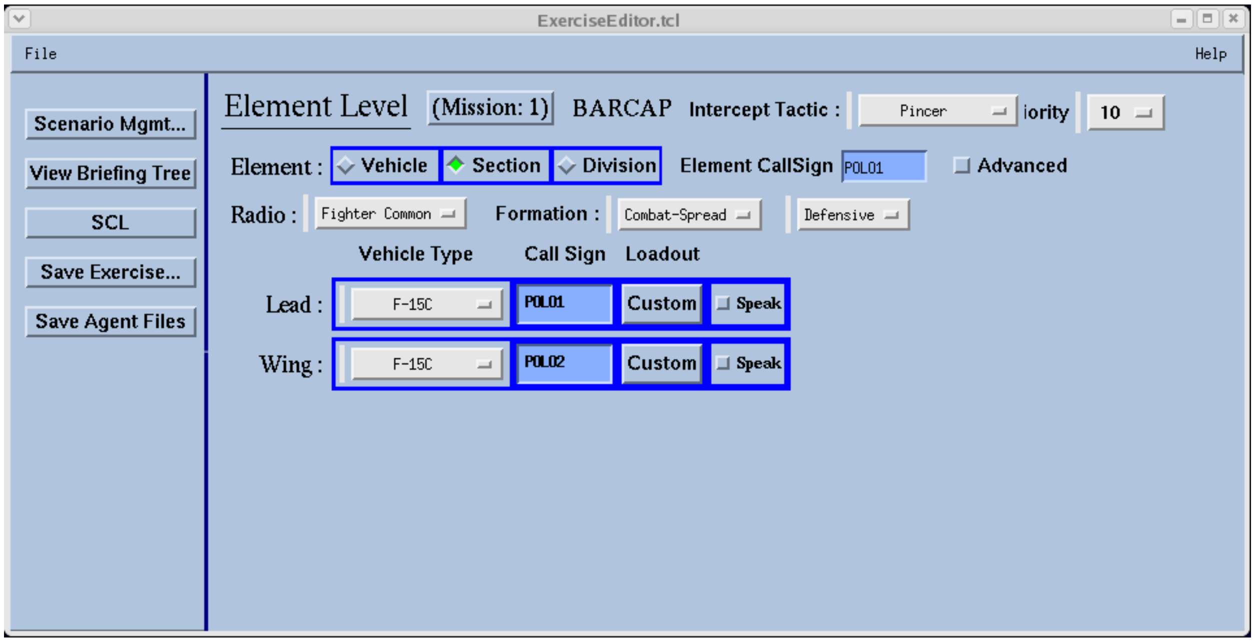
Figure 22. Exercise Editor - Mission Level

Here you will define a mission that may only contain a single element, though this may be a single, two, or four ship element. Begin by selecting a mission type from the dropdown menu at the top. Mostly Navy terms are listed there. Define a takeoff time for all aircraft in the mission. The aircraft will not actually all take off at once, but in sequence. Select the takeoff and landing base from the dropdowns, and you will notice these contain only the points you defined at the *Exercise* level. Enter the controller callsign (this can be the AWACS, if you create one). We did not involve tankers with our study. Note that you can navigate back up to the *Event* level by clicking the *Event 1* button at the screen top.

You may define CAP parameters (point, altitude, orientation, leg length) by clicking the *Cap Parameters* button. You may optionally change the predefined commit criteria by clicking the *Commit Criteria* button. If you have defined multiple points, you may create a *Route to CAP* by listing the points you desire in sequence. Again, the *Verify Mission* button can be used to error check once the elements are all created. The *Scheduled* checkbox will be green if the mission is scheduled, indicating it will takeoff. If this box is not checked, the mission will not fly. NOTE: This is the only way we found to remove a defined mission, as we found no way to delete a mission once created.

Finally, click on *Edit Mission Elements* to define the actual aircraft and parameters for this mission.

Element Editor



Element Level (Mission: 1) BARCAP Intercept Tactic : Pincer iority 10

Element : Vehicle [✓ Section] Division Element CallSign POL01 Advanced

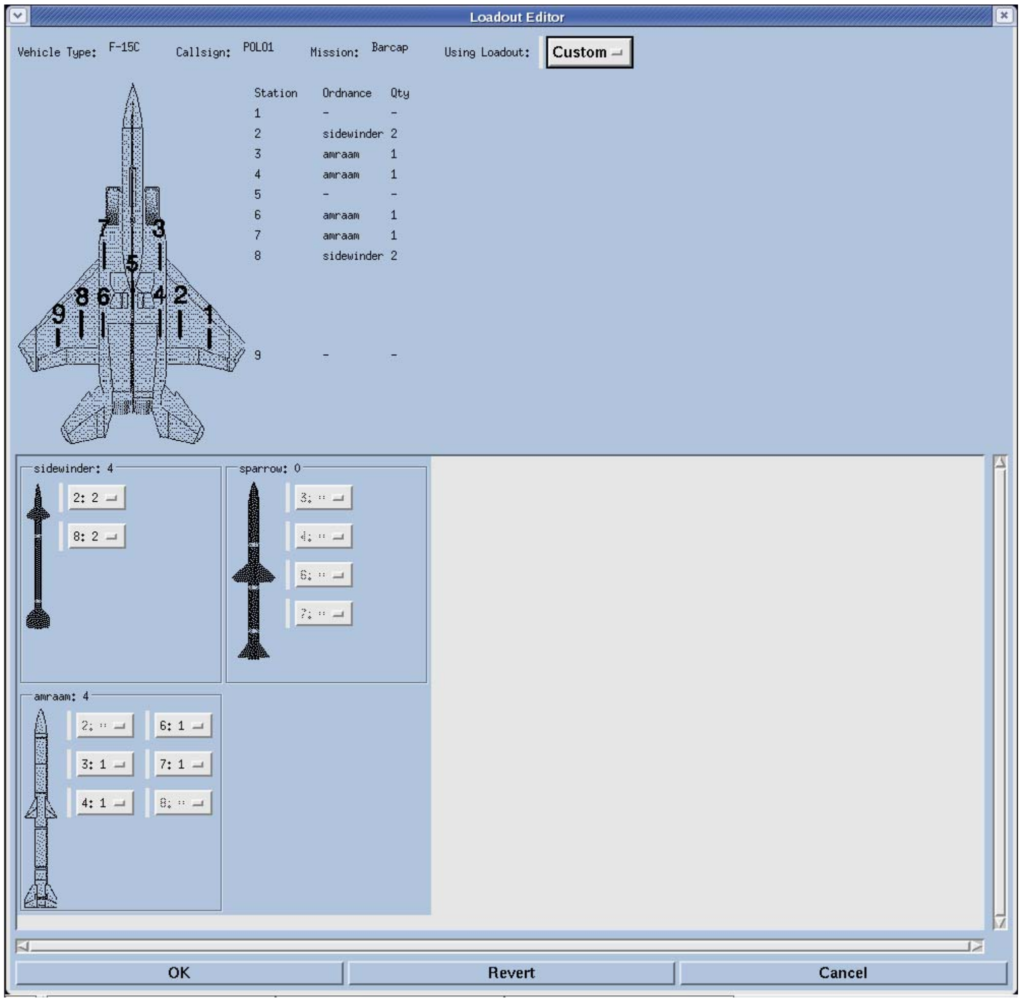
Radio : Fighter Common Formation : Combat-Spread Defensive

	Vehicle Type	Call Sign	Loadout	
Lead :	F-15C	POL01	Custom	Speak
Wing :	F-15C	POL02	Custom	Speak

Figure 23. Exercise Editor - Element Level

On this screen, you will select the element characteristics. Select *vehicle* (single ship), *section* (two-ship), or *division* (four-ship). Select the vehicle type from the drop-down menu. For the callsign, be sure **not** to use spaces if you plan to interact with EAAGLES, since you cannot define callsigns with spaces in that environment (with certain modules). Select the radio from the dropdown list. Note that it contains only the frequencies you defined at the *Exercise* level. The *speak* button activates *soar speak* on port 0, but is not operational in this release of JSAF.

You may define a custom loadout for your element, once you have selected an aircraft type. Click on the button in the *Loadout* column to bring up the loadout GUI. Not all the weapons stations and possible loadouts are accurate, but they provide a suitable cross-section of weapons appropriate for the missions we performed.



Vehicle Type: F-15C Callsign: POL01 Mission: Barcap Using Loadout: Custom

Station	Ordnance	Qty
1	-	-
2	sidewinder	2
3	amraam	1
4	amraam	1
5	-	-
6	amraam	1
7	amraam	1
8	sidewinder	2
9	-	-

sidewinder: 4 — 2: 2, 8: 2

sparrow: 0 — 3: '', 4: '', 6: '', 7: ''

amraam: 4 — 2: '', 6: 1, 3: 1, 7: 1, 4: 1, 8: ''

Figure 24. Soar Agent Loadout

Once a *custom* loadout has been defined, ensure that *custom* is selected on the elements screen in the loadout column. When you have completed creating the element, click on the *(Mission 1)* button to regress to the mission editor.

You may optionally verify the Mission and the Scenario as previously discussed. When you are satisfied, you must take two final steps to save the scenario and create the agents:

1. Return to the *Event* level and select *Save Scenario Files*. This action saves the scenario file and creates the agents for you from the precompiled binaries. After this point, the agents are not controllable or editable from within JSAF, though you may continue to edit them using the Exercise Editor.

2. Return to the *Exercise* level and select *File, Save Exercise*. This simply saves your exercise file for future editing, and has nothing to do with JSAF.

You may now close the Exercise Editor.

Running JSAF with TAS agents

JSAF can be run with a myriad of command-line options. In order to run JSAF with TAS agents, a script is provided that sets the majority of the options automatically for running the TAS agents. This script may be accessed by creating a terminal window, and browsing to the runscripts folder using the following command:

```
cd /usr/stow/build/JSAF/runscripts
```

The script can then be run using the following syntax (on one line):

```
./soarsf –ex blue1 –ex_path /usr/stow –terrain swus_tsga_6x6_v4_020812.dir
    –federation juo –ev 1 -nopo
```

The parameters for this script are:

-ex <exercise name> Whatever you have named your exercise. If you enter an exercise that has not yet been created, the script will error.

-ex_path <path_name> The path that contains your *exercises* folder you created in setup. If you followed the instructions in this paper, that value will be */usr/stow*.

-terrain <terrain folder> The terrain file you plan to use with this exercise. This is usually a small section of the globe, in detail. JSAF will also load the *world_thinned database* so that your JSAF simulation will have global coverage.

-federation <federation> There are several federations available, *juo* is the one that works best for stable HLA communication at the facilities utilized for this paper.

-ev <event number> Most of the time you can simply list event 1, regardless of how many events you have created.

-nopo – Stands for "non-persistent objects" and is required so that JSAF ignores any ghost tracks that it sees from dead entities. DIS gateways tend to ghost entities long after they have been eliminated.

Depending upon the speed and configuration of your computer, it may take several minutes to load JSAF. Once it is loaded, you will have no entities or scenarios running, only the JSAF environment. You may move your view to different locations around the world by doubleclicking on the *Cursor Point Coordinates* window (see Figure 25. JSAF Screen Capture), and then manually entering new coordinates to center your screen.

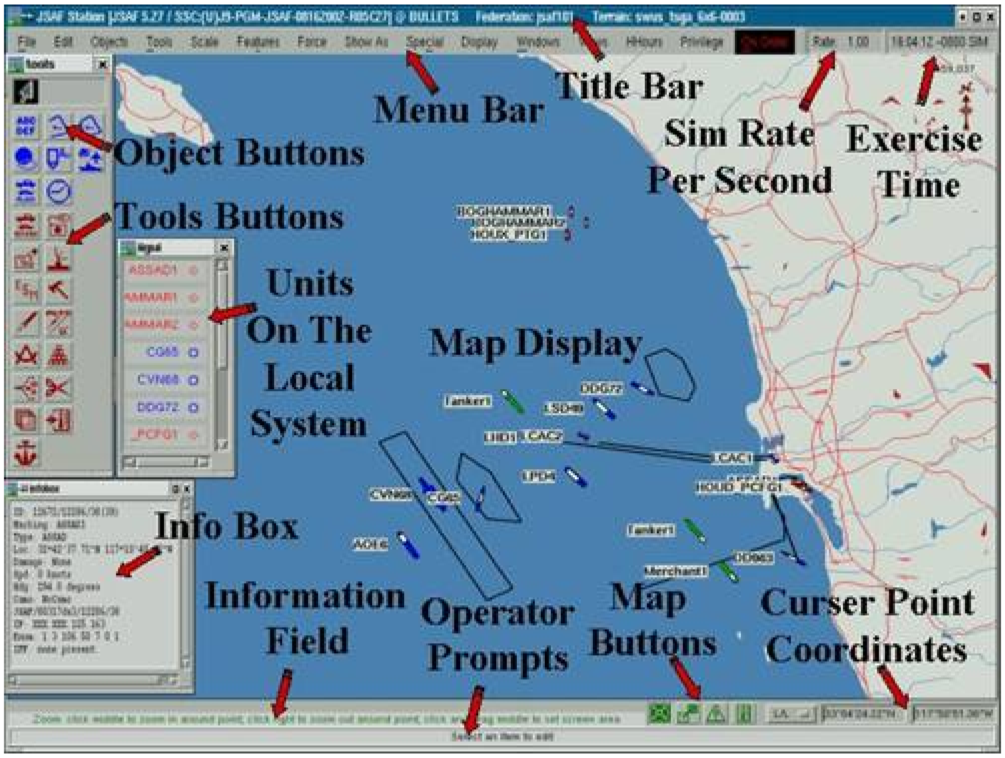
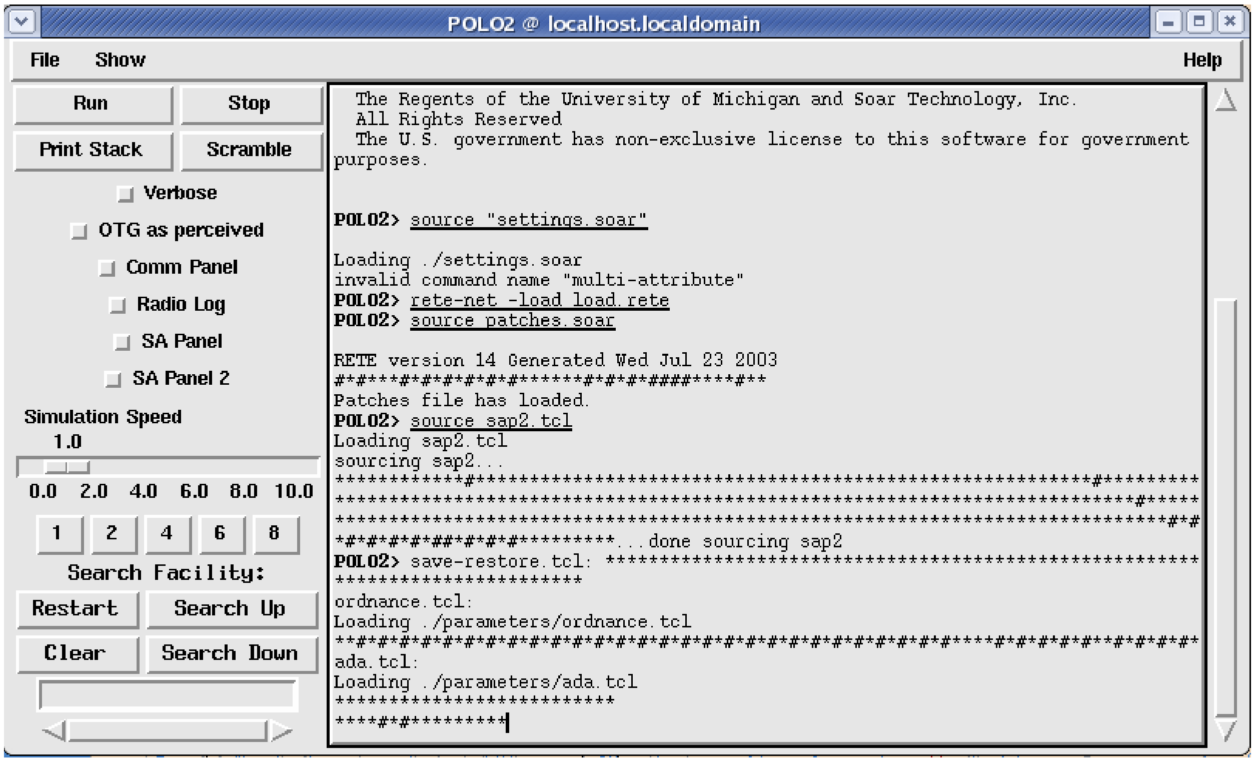


Figure 25. JSAF Screen Capture

This guide will not go into great depth into how to create and control entities in JSAF, as there is an extensive amount available in the “JSAF USER’S GUIDE” that is included in the CD with this paper. Non-TAS entities such as aircraft and surface air defense batteries can easily be created to attack and be attacked by TAS agents. They can be easily controlled through the JSAF interface, unlike the TAS agents.

Loading the TAS agent scenarios

Once your location is set where you want, you may now load your previously created scenario to load the TAS agents. Do so by selecting *File*, *Load Scenario* and select the scenario you created. Usually it is called “scenario.dat.” Disregard the similarly-named filed “scenario.dat.*xx*” as these are old versions of your scenario. Exercise Editor doesn’t erase old versions, it archives them by placing a number after the filename. Once you have loaded your scenario, you will see a window load for each agent:



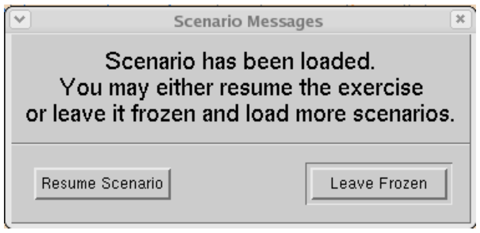
```
  The Regents of the University of Michigan and Soar Technology, Inc.
  All Rights Reserved
  The U.S. government has non-exclusive license to this software for government
purposes.


POLO2> source "settings.soar"

Loading ./settings.soar
invalid command name "multi-attribute"
POLO2> rete-net -load load.rete
POLO2> source patches.soar

RETE version 14 Generated Wed Jul 23 2003
#*#***#*#*#*#*#*#*******#*#*####*****#**
Patches file has loaded.
POLO2> source sap2.tcl
Loading sap2.tcl
sourcing sap2...
************#*********************************************************#*********
**************************************************************************#*****
*****************************************************************************#*#
*#*#*#*#*##*#*#*#**********...done sourcing sap2
POLO2> save-restore.tcl: *******************************************************
***********************
ordnance.tcl:
Loading ./parameters/ordnance.tcl
**#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#****#*#*#*#*#**#*#*#*#*
ada.tcl:
Loading ./parameters/ada.tcl
**************************
****#*#*********
```

Figure 26. Soar Agent Window

From this panel you can access all of the TAS elements and interfaces for each of the agents. If you have many agents, you will have many of these screens, using the multiple workspaces in Linux is an excellent way to manage all these agents.

Once the agents have loaded, you must click on the *Resume Simulation* button to begin the scenario and unfreeze the clock.

You may view the decisions made by and the situational awareness of each agent through this panel, as well as (somewhat) interact with the agents through several



Scenario has been loaded.
You may either resume the exercise
or leave it frozen and load more scenarios.

Figure 27. Scenario Messages

modules accessible from the agent panel.

Situational Awareness Panel (SA Panel 2)

To open the Situational Awareness Panel, click on the *SA Panel 2* button. Avoid the *SA Panel* button as it is an obsolete version of the same panel, with far less functionality. The following window will open independent of JSAF after a few seconds:

The owning aircraft of the SAP will always be in the center of the screen.

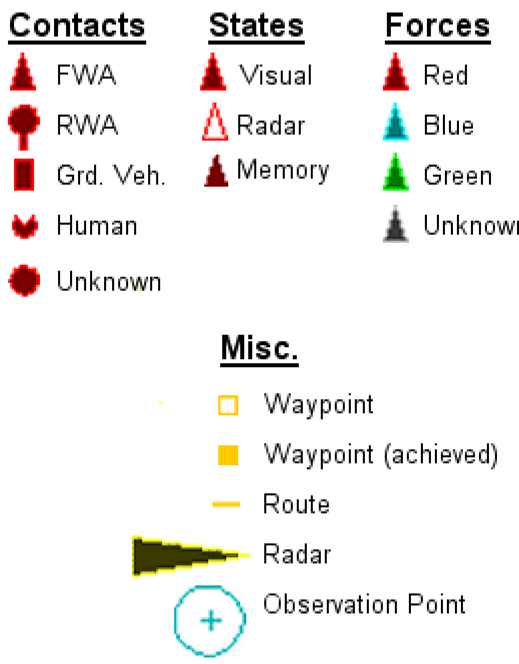
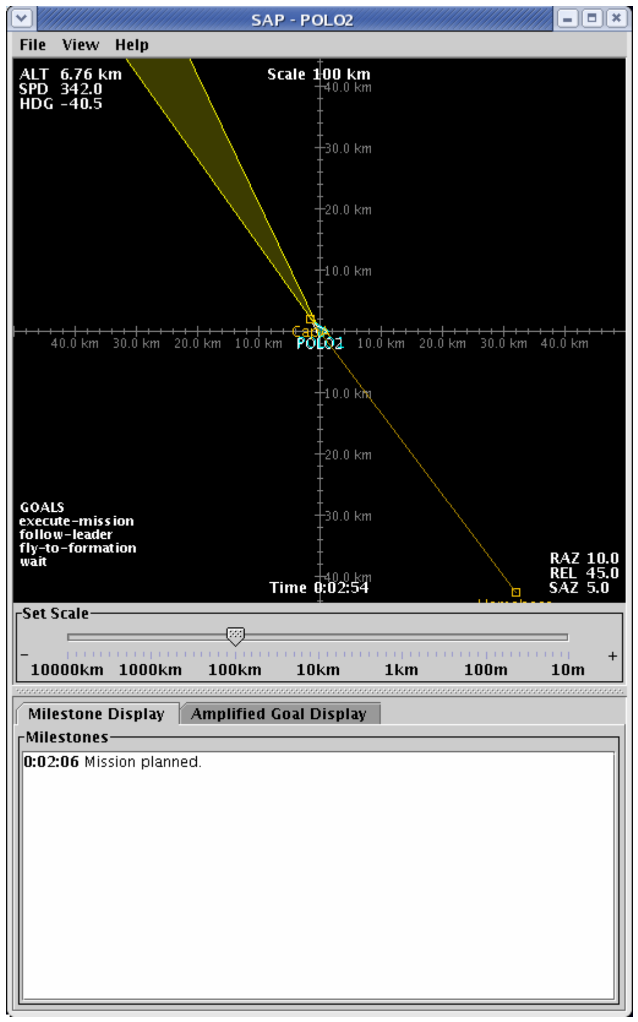


Figure 28. SAP Legend



ALT 6.76 km
SPD 342.0
HDG -40.5

Scale 100 km

GOALS
execute-mission
follow-leader
fly-to-formation
wait

Time 0:02:54

RAZ 10.0
REL 45.0
SAZ 5.0

0:02:06 Mission planned.

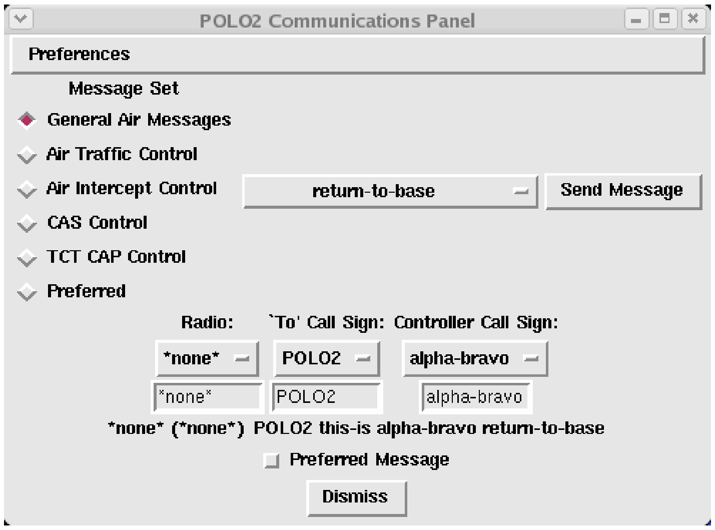
Figure 29. SAP Screenshot

This legend explains the basics of what you see on the SAP. Note: it is very difficult to tell the difference between a Visual and a Memory track.

There are a few significant options on the SAP. First, you can record to a logfile for replay on any Java-supporting system (Windows included). Next, you can select "Amplified Goal Display" which will indicate the rules the agent is currently following. Lastly, you can select different agents to view on a single SAP, without having to open an instance of the SAP for each agent you want to monitor.

Communications Panel

The comm. panel is your only means of communicating with TAS agents. The list of commands is somewhat limited, and mostly Navy comm. There is a great deal of "navigation" comm. ability, but very little tactical ability. Due to the lack of functionality of the SoarSpeak module, this presents an inability to target or order a wingman in the tactical environment. Due to the Soar agent decision-making process, the agents may or may not elect to follow directions given to them by their designated controllers.



Message Set: [✓ General Air Messages] Air Traffic Control, Air Intercept Control, CAS Control, TCT CAP Control, Preferred

return-to-base

Radio: *none* `To' Call Sign: POLO2 Controller Call Sign: alpha-bravo

none (*none*) POLO2 this-is alpha-bravo return-to-base

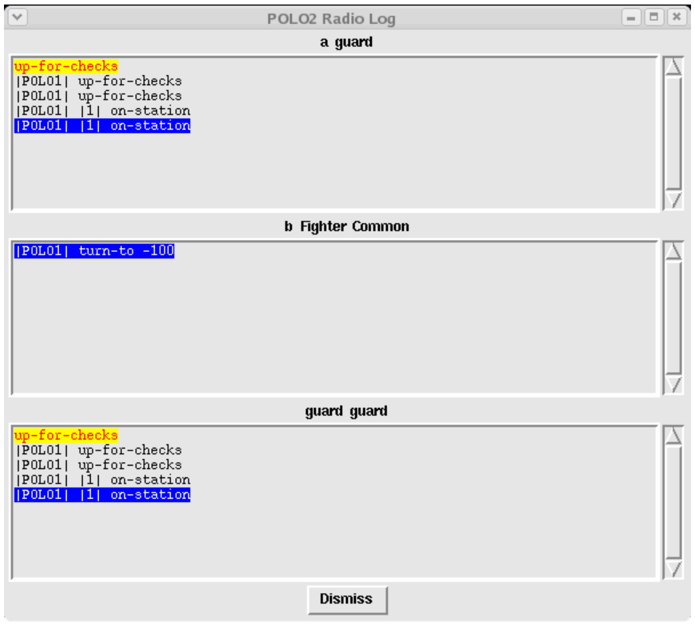
☐ Preferred Message

Figure 30. Soar Agent Communication Panel

To use the comm. panel, select the type of communication with the radio buttons to the left, select a predefined message from the drop-down menu, fill in any required parameters, select a radio and the agent you are calling, then hit *Send Message* to send. You can observe your radio call going out and any response from the agent via the *Radio Log Panel*.

Radio Log Panel

The radio log panel is the receiving end of your communication with TAS agents. It also allows you to observe the comm. going on between agents. There are no controls available, it is monitor only.



a guard

```
up-for-checks
|POL01| up-for-checks
|POL01| up-for-checks
|POL01| |1| on-station
|POL01| |1| on-station
```

b Fighter Common

```
|POL01| turn-to -100
```

guard guard

```
up-for-checks
|POL01| up-for-checks
|POL01| up-for-checks
|POL01| |1| on-station
|POL01| |1| on-station
```

Figure 31. Soar Agent Radio Log

DIS Gateway

In order to integrate with the EAAGLES environment, it is necessary to have a functioning DIS gateway. If you have several JSAF computers working together, they are connected by HLA, and only one of those computers needs to be running the DIS gateway.

If you have only the one machine running JSAF, then it must run JSAF, the TAS code, and the DIS gateway, which could cause slow output. Slow output to the DIS gateway makes the agent seem "jerky" to the pilot in the "human-in-the-loop" simulator. DIS performs "dead reckoning" of its data on each entity between times of receiving data from the entities source. Therefore, if an entity is maneuvering, but the entity's computer is overloaded and slow to transmit data, the entity's representation to the human in the simulator is unrealistic, annoying, and inconvenient. If you plan to run several TAS agents and entities on a single computer, it is desirable to run the Gateway on a separate JSAF computer.

The procedures for running the DIS Gateway are extremely simple, but important. The Gateway may be accessed by creating a terminal window, and browsing to the Gateway folder using the following command:

```
cd /usr/stow/build/JSAF/src/Gateway/
```

The program can then be run using the following syntax (on one line):

```
./gateway -exercise blue1 –terrain
        world_thin_fmt8_7_180W180E75S75N_v01 -dis204 -udp
        -disport 3000 -federation juo
```

The parameters for this script are:

-exercise <exercise name> Whatever you have named your exercise. If you name an exercise that has not yet been created, the script will error.

-terrain <terrain folder> You should load the *world_thinned database* so that your DIS Gateway will have global coverage. The gateway's visibility is constrained by the lateral limits of the terrain coverage.

-federation <federation> There are several federations available, *juo* is the one that works best for stable HLA communication at the facilities utilized for this paper. The Gateway will only transmit/receive data to/from this federation.

-udp Specifies network transmission type

-dis<version> Specifies what version of DIS to translate the data to. The EAAGLES network uses DIS version 204. DIS monitoring software is available at AFRL to oversee up to version 204. If you do not enter this parameter, the Gateway defaults to version 205.

-disport <port number> Specifies the port number to use for DIS transmission. The JSAF Gateway defaults to port 3005 if this is not specified. The EAAGLES network at AFRL/HED utilizes port 3000.

Appendix C: Installation CD File List

Compact Disk 1

Fedora Core Installation Disk 1

Notes: This is a bootable CD. You must set your BIOS to “Boot from CD” to begin Linux installation.

Rights to the Fedora Core 3 Operating System belong to Red Hat, Inc. This software is open source and downloadable from:

http://fedora.redhat.com/

Compact Disk 2

Fedora Core 3 Installation Disk 2

Compact Disk 3

Fedora Core 3 Installation Disk 3

Compact Disk 4

Fedora Core 3 Installation Disk 4

Compact Disk 5

JSAF Installation CD

File	Size
jdk-1_5_0_03-linux-i586-rpm.bin	46959KB
jre-1_5_0_02-linux-i586-rpm.bin	16007KB
JSAFUsersGuide.zip	67964KB
SoarTech-VISTA-1.0.zip	3651KB
yum.conf	8KB

Directory: /JSAF_CD/

File	Size
ChangeLog	19KB
ExecutableInstall.sh	7KB
external.tgz	18358KB
LinkDirs.sh	5KB
MD5SUM	1KB
patches.soar	16KB
README.txt	4KB
Release_Notes	7KB
SourceInstall.sh	7KB
SSC_U_JSAF_2004_linux_g++-3.2_Release_24DEC2004.tgz	154640KB
SSC_U_JSAF_2004_linux_g++-3.3_Release_24DEC2004.tgz	203092KB
SSC_U_JSAF_2004_linux_g++-3.4_Release_24DEC2004.tgz	152527KB
SSC_U_JSAF_2004_src_Release_24DEC2004.tgz	59994KB

Notes: *patches.soar* is the only file modified from the JFCOM download. It incorporates a fix for the TAS agents to allow JSAF to read their weapons loadouts. This fix is discussed previously in this paper.

Compact Disk 6

Directory: /TERRAIN_CD/

File	Size
jak_juo_041101_v09_ctdl_fmt8_7.tgz	231059KB
swus_tsga_6x6_v4_020812_ctdl_fmt8_7.tgz	21936KB
TerrainInstall.sh	6KB
world_thin_fmt8_7_180W180E75S75N_v01.dir.tgz	441516KB

Notes: *swus_tsga_6x6_v4_020812_ctdl_fmt8_7.tgz* converted specifically for this project to overlap with the terrain used by AFRL/HECP's EAAGLES simulator.

Appendix D: Contacts List

AFRL (US Air Force Research Laboratory, Warfighter Training Division, Mesa, AZ)
  Boyle, Garry (Simulation)
    e-mail: garry.boyle@mesa.afmc.af.mil
  Smoot, Don
    Lockheed Martin Systems Management
    phone: Office: 480-988-9773 ext 418
      Cell: 480-227-4289
    e-mail: Don.Smoot@mesa.afmc.af.mil

Chechio, Mark
  JSAF Operator Course (Pricing: $2,500 per student)
  e-mail: training@bmh.com
  phone: (757) 857-5670 x212 (for group discount rates or other information)
  web page: http://www.bmh.com/bmhinternet/services/jsafOper.htm

FTP Sites
  ftp2.soartech.com – Use email address as username
  trojan.spawar.navy.mil – See Dr. Miller for access

Gardner, Nancy K.
  U.S. Army Engineer Research and Development Center (ERDC-TEC-VA)
  e-mail : Nancy.K.Gardner@erdc.usace.army.mil
  web pages: http://mel.dmso.mil , http://www.erdc.usace.army.mil/

Haas, Mike
  US Air Force Research Laboratory, Human Effectiveness Division (AFRL/HECP)
  phone: DSN 785-8768
  e-mail: Michael.Haas@wpafb.af.mil

Kerr, Kris
  SOAR Expert at JFCOM
  e-mail: Kris.Kerr@je.jfcom.mil

McNally, Brooke.
  USAF Simulation and Analysis Facility (SIMAF) (ASC/HPMT)

e-mail: Brooke.McNally@wpafb.af.mil
phone: DSN 785-0626

Meckstroth, Gregory L.
JFCOM's primary JSAF developer. Bottom line guy for all installation issues.
address: SPAWARSYSCEN 246213
53140 Systems St Rm 317
San Diego CA 92152-7566
e-mails: greg@spawar.navy.mil
jsaf5-support@lads.is.lmco.com – Primary support request address
phone: Office: 619-553-2117
Mobile: 619-341-0901
Fax: 619-553-6902

Menke, Timothy E.
USAF Simulation and Analysis Facility (SIMAF) (ASC/HPMT)
e-mail: Timothy.Menke@wpafb.af.mil
phone: DSN 785-1286

Soar Technology, Inc.
Nielsen, Paul
Vice President & Senior Scientist
address: 3600 Green Court, Suite 600
Ann Arbor, MI 48105
e-mail: nielsen@soartech.com
phone: Office: 734-327-8000 ext 201
Cell: 734-330-6177
web page: www.soartech.com
Wallace, Al
Chief Operating Officer
e-mail: wallace@soartech.com
phone: Office: 734-327-8000 ext 218
Cell: 734-652-0173
Fax: 734-913-8537
Koss, Frank
CIO & Senior Systems Engineer
e-mail: koss@soartech.com
phone: Office 734-327-8000 ext 216

Bibliography

"Enhanced Air-to-Air & Air-to-Ground Linked Environment Simulation (EAAGLES) and EAAGLES Toolkit." AF Simulation and Analysis Facility Bullet Background Paper. Nov 2004.

"Enhanced Air-to-Air & Air-to-Ground Linked Environment Simulation (EAAGLES) and EAAGLES Toolkit." AF Simulation and Analysis Facility Talking Paper. Dec 2004.

"History." Soar Technology, Inc. corporate web page. 07 April 2005. http://www.soartech.com/company.history.php .

"Joint Semi-Automated Forces." Abstract hosted by the Navy Modeling and Simulation Management Office. 09 Feb 1998. http://navmsmo.hq.navy.mil/index.cfm?RID=MNS_N_1010133 .

Joint Semi-Automated Forces (JSAF) User's Guide. Volume I: Introduction and Basic Controls. 17 Jun 2003.

"Joint Semi-Automated Forces (JSAF) / Synthetic Theater of War (STOW)." Abstract hosted by the Human Performance Center. 01 Mar 2005. https://www.spider.hpc.navy.mil/index.cfm?RID=TTE_OT_1000011 .

"Joint Semi-Automated Forces." Abstract hosted by the Air Force Modeling and Simulation Resource Repository. 7 April 2005. http://afmsrr.afams.af.mil/index.cfm?RID=MDL_AF_1000066 .

Kalus, Tony and Frank Ritter. "An Introduction to the Soar Cognitive Architecture." University of Portsmouth, UK and Penn State Powerpoint Presentation. 2 April 2003. http://acs.ist.psu.edu/papers/pst14/bamberg-soar.ppt .

Laird, John E. and Clare Bates Congdon. *The Soar User's Manual* (Version 8.5). University of Michigan. June, 2004.

Lehman, Jill Fain, John Laird and Paul Rosenbloom. "A Gentle Introduction to Soar, an Architecture for Human Cognition." University of Michigan Soar Homepage. 04 Apr 2005. http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/Gentle.pdf.

Levine, Randyll and Timothy Menke. "EAAGLES – Virtual Simulation," *ASC MSRC Journal*: 20-21 (Fall, 2003)

Menke, Tim. “New Modeling and Simulation Development: A Focused Effort To Bridge Development: A Focused Effort To Bridge The Gap Between The Developmental The Gap Between The Developmental Acquisition And Test Communities” Presentation to Software Technology Conference at Salt Lake City, Utah. 01 May 2003.

Nielson, Paul, Don Smoot, and JD Dennison. “Participation of TacAir-Soar in RoadRunner and Coyote Exercises at Air Force Research Lab, Mesa AZ.” Unpublished report from Soar Technologies, Inc. website. n. pag. http://www.soartech.com/pubs/rr-coyote.pdf . 08 April 2005.

Trevisani, Dawn A. *Command, Control, Communications, Computer, Intelligence, Surveillance And Reconnaissance (C4ISR) Modeling And Simulation Using Joint Semi-Automated Forces (JSAF).* Final Technical Report. Northrup Grumman, Inc. AFRL-IF-RS-TR-2003-144. June 2003.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 03/06/2005	**2. REPORT TYPE** Graduate Research Project	**3. DATES COVERED** *(From – To)* January 2005 – June 2005

4. TITLE AND SUBTITLE **DEVELOPING A VALIDATION METHODOLOGY FOR TACAIR SOAR AGENTS IN EAAGLES**	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Alford, Lewis E. III, Maj, USAF; Dudas, Brian A., Maj, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, Building 640, WPAFB OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GOS/ENS/05-01
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Eileen A. Bjorkman, Col, USAF; Deputy Director, Capabilities Integration Directorate; ASC/XR; DSN 785-0934, (937) 255-0934	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Air Force Simulation and Analysis Facility (SIMAF) at Wright Patterson AFB, OH is currently conducting an effort to integrate TacAir Soar agents into the Enhanced Air-to-air and Air-to-Ground Linked Environment Simulation (EAAGLES) environment. SIMAF plans to support customizable agents in the Soar interpretive language, compilable for use in EAAGLES. TacAir Soar (TAS) agents are currently only utilized in the Joint Semi-Automated Forces (JSAF) environment, but have potential for use in EAAGLES. SIMAF requested research be conducted on a validation methodology to apply to the agents' behavior once they have been successfully imported into the EAAGLES environment. This methodology developed could then be used on all variants of the final version of the agents, once delivered by Soar Technologies, Inc. This research project investigated the possibility of "verification by comparison" with the TacAir Soar agent's current behavior and performance in JSAF while integrated with "human in the loop" simulators. By comparing the new models to the already validated models in JSAF, the desire was to eliminate the need for a "ground up" verification. This paper outlines the steps taken to successfully construct a working JSAF/TacAir Soar model, as well as illustrates the deficiencies and limitations encountered. Finally, recommendations are made as to the future development of validation/verification methodology for SIMAF's efforts.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF: UNCLASSIFIED			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. J.O. Miller
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	102	(937) 255-6565, ext 4326 (john.miller@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18